Université de Montréal

Turbulence de surface pour des simulations de fluides basés sur un système de particules

par
Cynthia Beauchemin

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Septembre, 2016

Résumé

En simulation de fluides, il est très difficile d'obtenir des simulations contenant un haut niveau de détails efficacement dû à la complexité des phénomènes étudiés. Beaucoup de travaux se sont attaqués à ce problème afin de développer de nouvelles techniques permettant d'augmenter la résolution apparente des fluides à plus faibles coûts de calcul. Les nouvelles méthodes adaptatives ou multi-échelles ont permis de grandement améliorer la qualité visuelle des simulations de fumée et de liquides, mais certains problèmes demeurent toujours ouverts et au centre de nombreuses recherches.

L'objectif de ce mémoire est d'élaborer une méthode multi-échelles afin d'augmenter la résolution apparente d'une simulation de liquide basée sur un système de particules déjà existante, un type de simulation très populaire grâce à ses propriétés de conservation d'énergie. Une telle méthode permettrait d'obtenir des simulations de résolution apparente élevée à bien moindres coûts de calcul et permettrait ainsi aux artistes d'obtenir un aperçu de leur simulation plus rapidement, tout en ayant un résultat de haute qualité. Nous présentons une méthode permettant de reconstruire une surface d'une telle simulation qui soit encline à la simulation de dynamique de surface afin d'injecter des détails de hautes fréquences occasionnés par la tension de surface. Notre méthode détecte les endroits sous-résolus de la simulation et y injecte de la turbulence grâce à de multiples oscillateurs à différentes fréquences. Les vagues à hautes fréquences injectées sont alors propagées à l'aide d'une simulation d'onde sur la surface. Notre méthode s'applique totalement en tant que post-traitement et préserve ainsi entièrement le comportement général de la simulation d'entrée tout en augmentant nettement la résolution apparente de la surface de celle-ci.

Mots clés : Simulation de liquide, turbulence, reconstruction de surface

Abstract

Accurately simulating the behaviour of fluids remains a difficult problem in computer graphics, and performing these simulations at a high level of detail is particularly challenging due to the complexity of the underlying dynamics of a fluid. A recent and significant body of work targets this problem by trying to augment the apparent resolution of an underlying, lower-resolution simulation, instead of performing a more costly simulation at the full-resolution. Adaptive or multi-scale methods in this area have proven successful for simulations of smoke and liquids, but no comprehensive solution exists.

The goal of this thesis is to devise a new multi-scale detail-augmentation technique suitable for application atop existing particle-based fluid simulators. Particle simulations of fluid dynamics are a popular, heavily-used alternative to grid-based simulations due to their ability to better preserve energy, and no detail-augmentation techniques have been devised for this class of simulator. As such, our work would permit digital artists to perform more efficient lower-resolution particle simulations of a liquid, and then layer-on a detailed secondary simulation at a negligible cost.

To do so, we present a method for reconstructing the surface of a liquid, during the particle simulation, in a manner that is amenable to high-frequency detail injection due to higher-resolution surface tension effects. Our technique detects potentially under-resolved regions on the initial simulation and synthesizes turbulent dynamics with novel multi-frequency oscillators. These dynamics result in a high-frequency wave simulation that is propagated over the (reconstructed) liquid surface.

Our algorithm can be applied as a post-process, completely independent of the underlying simulation code, and so it is trivial to integrate in an existing 3D digital content creation pipeline.

Keywords: liquid simulation, turbulence, surface reconstruction

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Je tiens d'abord à remercier mon directeur de recherche, Derek Nowrouzezahrai, pour m'avoir transmis sa passion lors de mon premier cours en informatique graphique au cours de mon baccalauréat, puis de m'avoir accepté comme étudiante à la maîtrise. Je veux particulièrement le remercier pour son soutien, sa patience, sa façon d'enseigner avec passion et son soucis de toujours veiller au bien de ses étudiants. Merci à Pierre Poulin, professeur au LIGUM, non seulement pour son enseignement mais également pour avoir été très présent au laboratoire afin de discuter, de nous aider et d'ajouter son unique touche d'humour. Merci à tous les étudiants du LIGUM : Adrien, Alexandre, Antoine, Arnaud, Aude, Chaitanya, Cihan, Dabid, David, Dorian, Étienne, Florian, Gilles-Philippe, Guofu, Jean-Philippe, Joël, Jonathan, Laurent, Marc-Antoine, Mélino, Olivier, Sonia et Yangyang pour l'entraide et les discussions, pour les séances de mots croisés et de jonglerie, pour le partage, pour les sorties et, de manière générale, pour avoir fait de mon environnement d'étude un endroit agréable, motivant et plein d'humour. Un merci particulier à mon partenaire de recherche Olivier Mercier avec qui ce fut un plaisir de travailler. Merci à ma famille, mes parents Michel et Colette et ma soeur Jeanne, pour m'avoir toujours soutenu et cru en moi. Merci à Eric Bourque et Autodesk, pour leur confiance et leur support. Merci au CRSNG pour la bourse de recherche à la maîtrise. Ce soutien financier m'a permis de me consacrer pleinement à ma recherche et mes études.

Chapitre 1

Introduction

Avec la demande toujours croissante d'effets visuels de haute qualité dans l'industrie du film et des jeux vidéo, il est impératif que les effets visuels soient des plus convaincants. Les rendus d'images, les modèles 3D, les animations et les simulations doivent représenter fidèlement la réalité, suffisamment bien pour leurrer le spectateur en lui laissant croire qu'il s'agit d'images capturées du monde réel. L'image suivante en est un exemple.

Figure 1.1 – Simulation tirée du film *Pirates des Caraïbes : Le Secret du coffre maudit*. ©Disney

Dans ce présent mémoire, nous nous intéressons plus particulièrement aux simulations de fluides de type liquide (par opposition aux fluides de type fumée ou feu) de haute qualité. Ceux-ci doivent se comporter comme le ferait un véritable fluide et contenir un haut niveau de détails. Le problème est que ces phénomènes sont extrêmement complexes et résultent de l'interaction d'une quantité astronomique de molécules. Il est évidemment impensable de simuler les interactions à cette échelle ; une seule mole d'eau (environ 18mL à 4° C) contient environ 2.01×10^{23} molécules d'eau. Il nous faut donc avoir recours à d'autres stratégies afin d'approximer ces phénomènes. Plusieurs méthodes ont été élaborées afin de simuler le comportement des liquides. Toutes ces méthodes échantillonnent les fluides (par

le biais de différentes méthodes) afin de connaître son état courant et arriver à approximer le mieux possible son état futur après un temps donné. À basses résolutions (peu d'échantillons), ces simulations donnent généralement de bien piètres résultats. Celles-ci s'améliorent effectivement en augmentant leur résolution (davantage d'échantillons), mais deviennent rapidement très coûteuses en calcul et en espace mémoire. De plus, il est très difficile par exemple pour des artistes en animation de fluides de manipuler des simulations lorsque celles-ci prennent plusieurs secondes, minutes, voire même heures afin de calculer leur comportement et en donner un aperçu pour une seule image.

Afin d'accélérer les calculs, certaines techniques ont opté pour des méthodes adaptatives. Ces méthodes tentent d'adapter la résolution spatiale et/ou temporelle des simulations en fonction du besoin de capture en fréquences pour une zone et/ou un intervalle de temps donné. Par exemple, les mouvements de l'eau dans les fonds marins sont généralement peu élevés en fréquences ; l'eau tend à en quelque sorte à se déplacer plus "globalement" et il n'est souvent pas nécessaire de l'échantillonner abondamment afin de simuler son comportement. À l'inverse, la surface d'un liquide peut présenter un haut contenu en fréquences (éclaboussures, vagues capillaires, etc.) et nécessiter un échantillonnage dense afin de capturer et de bien simuler tous les détails visibles.

D'autres méthodes utilisent quant à elles des modèles multi-échelles. Elles vont par exemple consécutivement faire plusieurs simulations à différentes échelles, de la plus basse à la plus haute. Cela a comme avantage de pouvoir rapidement avoir un aperçu d'un fluide tout en obtenant un résultat final qui soit à la fois de haute qualité et fidèle à l'aperçu. Ces méthodes se basent sur l'hypothèse qu'à partir d'un certain point, les fréquences suffisamment élevées n'ont plus d'influence sur les fréquences plus basses. Ceci n'est pas vrai, d'autant plus qu'avec la dissipation, les hautes fréquences diminuent en fréquences, mais reste une supposition raisonnable. Par exemple, il est raisonnable de supposer qu'une goûte d'eau qui tombe à la surface de la mer ne devrait pas déclencher une vague capable de faire chavirer un bateau.

Ce mémoire présente une nouvelle méthode multi-échelle afin d'ajouter de la *turbulence* et des détails de surface à une simulation de liquide basée sur un système de particules. Nous avons développé cette méthode dans le but d'améliorer l'apparence de simulations de fluides de type FLIP (section 2.1.3 et 3.1) car ce type de simulation est très populaire grâce à sa simplicité et à ses propriétés de

conservation d'énergie. Par contre, peu de travaux ont été réalisés afin d'augmenter la résolution apparente de simulation FLIP en post-traitement. Notre méthode a été conçue initialement pour des simulations FLIP de base, mais se généralise également trivialement à d'autres types de simulations lagrangiennes (basées sur un système de particules). Notre méthode a l'avantage de diminuer la complexité de la simulation puisqu'elle s'applique sur la *surface* d'un fluide (complexité quadratique) et non sur le *volume* du fluide (complexité cubique) et permet donc d'améliorer les détails à la surface à bien moindres coûts (voir Figure 1.2).

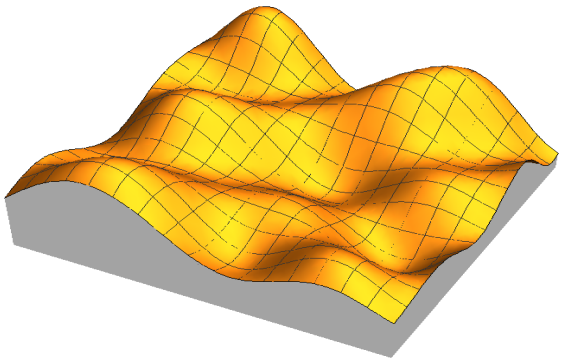


Figure 1.2 – En faisant une simulation sur la surface (en jaune), on diminue la complexité de la simulation par rapport à une simulation sur le volume (en gris). En effet, la surface constitue un sous-espace 2D du volume 3D et on obtient donc une complexité quadratique plutôt que cubique.

Nos contributions comprennent

- la génération robuste, spatialement et temporellement cohérente, d'une surface sans maillage et encline à des simulations de dynamique de surface,
- de nouvelles contraintes lisses qui assurent que la surface générée soit à la fois spatialement et temporellement cohérente et fidèle à la simulation sous-jacente,
- une méthode robuste afin de détecter les endroits sous-résolus de la simulation en entrée, et
- un nouvel opérateur laplacien applicable sur un ensemble de points représentant une surface.

Notre méthode d'augmentation de résolution apparente des fluides est présentée au chapitre 4 en la reproduction de l'article originalement produit en lien avec cette recherche, mais nous introduirons

d'abord le lecteur à des travaux antérieurs et au contexte technique nécessaire afin de faciliter la lecture et la compréhension de l'article. Au chapitre 2, nous survolerons l'état de l'art de la simulation de fluides (section 2.1), des méthodes de reconstruction et de suivi de surfaces de liquides (section 2.2) ainsi que des modèles de turbulence (section 2.3). Au chapitre 3, nous verrons plus en détail les éléments techniques préalables à la lecture de l'article.

Chapitre 2

État de l'art

Nous présentons dans ce chapitre un aperçu des méthodes représentant l'état de l'art dans le domaine de la simulation de fluides basée sur la physique, de la localisation de surface de fluides et de l'augmentation de résolution de simulation de fluides.

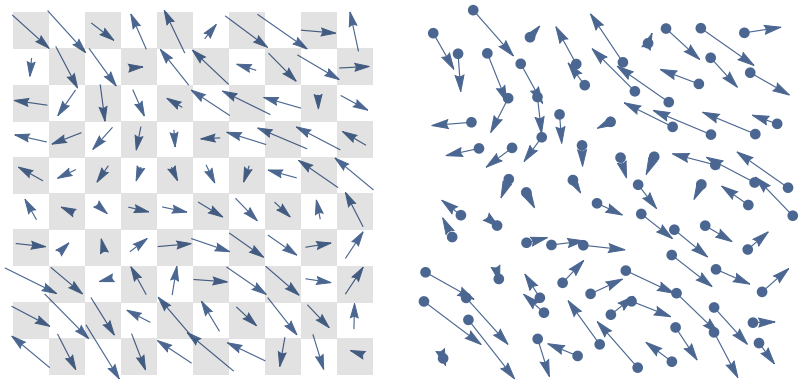
2.1 Simulations de fluides

Bien que notre travail ne porte pas spécifiquement sur les simulations de fluides, mais bien sur l'augmentation de la résolution de celles-ci, il est tout de même pertinent de connaître le contexte dans lequel nous agissons. Nous verrons donc ici un aperçu des grandes techniques utilisées en simulation de fluides.

Il existe deux grandes approches en simulation de fluides : eulériennes et lagrangiennes. Ces deux approches tentent de résoudre les équations de motions des fluides de Navier-Stokes en traitant le continuum des fluides de différentes manières.

2.1.1 Méthodes eulériennes

Les méthodes eulériennes utilisent généralement des grilles cartésiennes afin de représenter les fluides (voir la figure 2.1a). Chaque case de la grille contient des informations locales sur le fluide telles que la vélocité, la densité, la température, etc. Parmi les références importantes en simulation de fluides eulérienne, on peut notamment compter les travaux de Stam [Sta99] ainsi que Fedkiw et al. [FSJ01] afin de résoudre les phénomènes de fumée. Ces travaux décrivent comment discrétiser les équations de Navier-Stokes sur une grille pour simuler des phénomènes de fumée. Pour les simulations de fluides à plusieurs phases (par exemple eau-air), ces équations s'appliquent également (avec de légères modifications) mais il faut aussi pouvoir déterminer si une case est occupée par du liquide ou de l'air. Plusieurs méthodes ont été développées à ces fins. On peut notamment compter la méthode *Level set* (LSM) [OF03] dans laquelle chaque case contient sa distance signée à la surface la plus



(a) Approche eulérienne (b) Approche lagrangienne

Figure 2.1 – Les grandes approches en simulation de fluides. À gauche, les méthodes eulériennes utilisent des grilles cartésiennes afin de stocker les valeurs nécessaires à la simulation. À droite, les méthodes lagrangiennes utilisent un système de particules où celles-ci contiennent les valeurs de la simulation.

proche. Ceci permet d'aisément reconnaître si une case contient du liquide (distance négative) ou de l'air (distance positive). Par contre l'advection (ou transport) de ce champ de distance entraîne de sévères pertes en volume puisqu'il est nécessaire de lisser le champ et que cela élimine les détails de surface et les structures fines. Afin de pallier ce problème, Pilliod et Pucket [PP04] proposèrent une méthode de volume de fluide (VOF), où chaque case contient plutôt un volume de liquide ; 0 si la case est vide jusqu'à 1 si la case est pleine. Ceci permet d'assurer que le volume global (total) reste constant tout au cours de la simulation en redistribuant le volume perdu dans les cases à la surface.

L'un des avantages des simulations eulériennes est qu'elles permettent aisément la discrétisation des dérivées partielles spatiales de Navier-Stokes grâce à l'utilisation de grilles régulières. Ceci permet de calculer robustement et efficacement les forces dues à la viscosité interne ainsi qu'à la pression interne du fluide. L'autre avantage est qu'il est souvent trivial d'extraire la surface (si celle-ci n'est pas déjà explicitement donnée) puisque ces simulations viennent généralement avec les distances signées à la surface ou une représentation permettant aisément de reconstruire ces distances. Par exemple, dans LSM, la surface est simplement et évidemment l'ensemble des points tels que la distance à la surface est nulle, points pouvant être obtenus facilement par des méthodes telles que le *Marching Cubes* [LC87]. Par contre, ce modèle est particulièrement sujet à des problèmes de dissipations, que

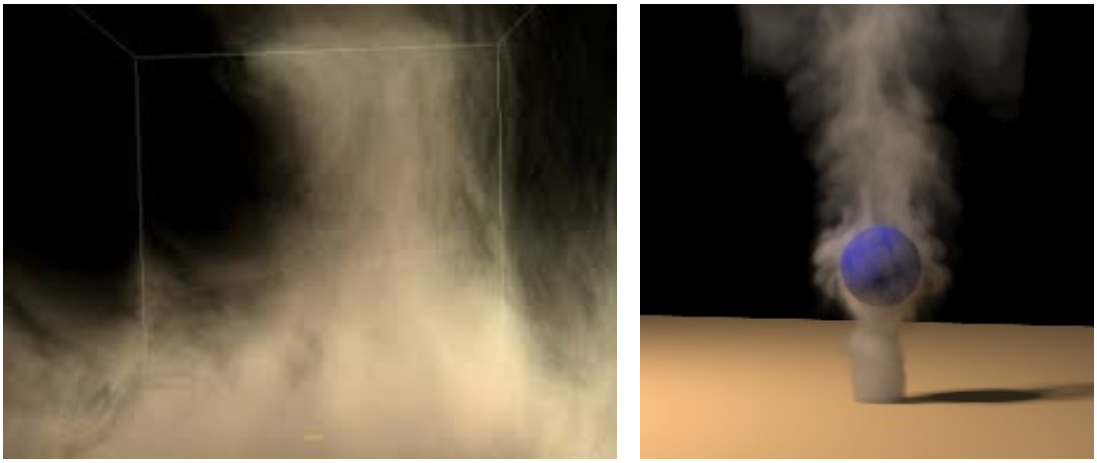


Figure 2.2 – Quelques résultats de simulations de fumée obtenus dans les travaux des Stam [Sta99] (à gauche) et Fedkiw at al. [FSJ01] (à droite).

ce soit en énergie et/ou parfois en volume puisqu'il faut régulièrement connaître des valeurs *entre les cases* lors de l'advection du fluide pour définir son nouvel état, nécessitant de moyenner les valeurs des cases voisines et entraînant donc inévitablement une sorte de lissage. Ceci se traduit visuellement par un fluide qui arrive trop rapidement à un état de repos ou qui perd des détails fins.

2.1.2 Méthodes lagrangiennes

Les méthodes lagrangiennes définissent le fluide comme un ensemble de particules interagissant entre elles, tel qu'illustré dans la figure 2.1b. Le modèle le plus largement utilisé est SPH (*Smoothed Particle Hydrodynamics*) [MCG03, MM13]. Chacune de ces particules contient une position, une vélocité, et optionnellement d'autres quantités (température, couleur, etc.). Tous les calculs sont faits directement sur l'ensemble des particules : le déplacement des particules (advection), les forces externes (gravité, etc.), les forces internes (viscosité, pression, etc.). Les principaux avantages des modèles lagrangiens sont qu'ils ne nécessitent aucune grille, donc n'ont aucune contrainte spatiale, ils ne perdent pas d'énergie lors de l'advection des particules contrairement aux modèles eulériens et permettent le traitement des collisions avec des obstacles arbitraires plus simplement. Bien que ce modèle semble plus naturel que le modèle eulérien, il possède aussi ses lacunes. En particulier, il est plus difficile de calculer des dérivées partielles spatiales sur un ensemble de particules se déplaçant

arbitrairement que sur une grille. Par exemple, il est beaucoup plus difficile de traiter correctement les forces dues à la pression interne dans ce type de système donnant souvent lieu à des erreurs dans les calculs garantissant l'incompressibilité du fluide. On pourra alors observer que le fluide se compresse et se décompresse, nuisant énormément au réalisme de la simulation.

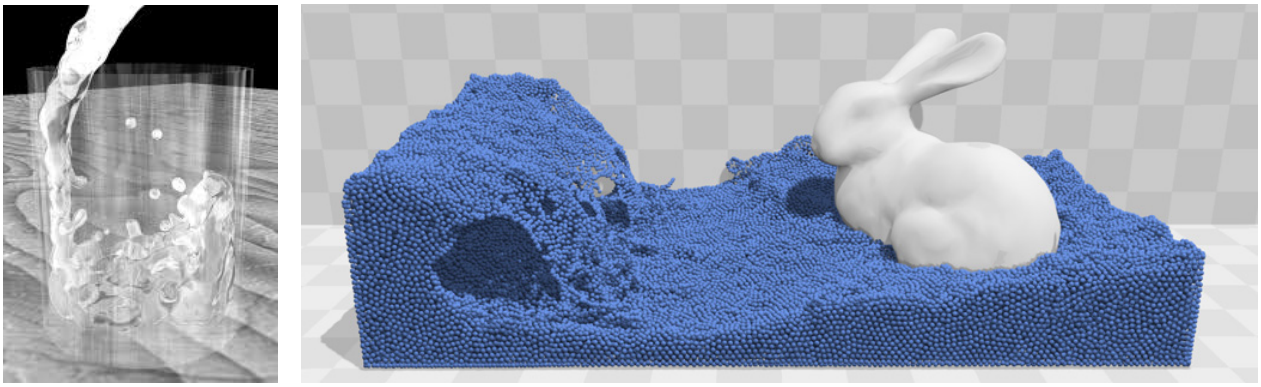


Figure 2.3 – Résultats de simulations SPH [MCG03, MM13].

2.1.3 Méthodes hybrides.

Les méthodes hybrides combinent quant à elles l'utilisation à la fois de particules et d'une grille. Une méthode de ce type largement utilisée aujourd'hui est la méthode PIC-FLIP [BR86] (*Particle-In-Cell, FLuid Implicit Particle*), mise de l'avant dans les récents travaux de Zhu et Bridson [ZB05]. L'idée de cette méthode est de combiner les avantages des simulations eulériennes et lagrangiennes dans une seule et même simulation. Plus spécifiquement, cette méthode utilise une grille afin de faire tous les calculs (viscosité, pression, etc.) *sauf* l'advection puisque c'est spécifiquement cette étape dans le modèle eulérien qui entraîne la dissipation de l'énergie. L'étape d'advection est entièrement vers l'avant (*forward*) faite à l'aide de particules, comme dans un modèle SPH.

Puisque ce sont les particules qui se déplacent en fonction du flux de la simulation, ce sont donc elles qui définissent où se situe le liquide, de la même manière que dans la méthode de marqueur particule [HW65]. Une case est considérée comme une case "liquide" si et seulement si elle contient au moins une particule. Une des différences majeures entre les particules de FLIP par rapport à SPH est que la pression est résolue sur une grille et non sur les particules entre elles. Cela se traduit par une

plus grande variance de densité de particules, qu'il est important de prendre en compte autant lors de la simulation que lors de la reconstruction de surface.

2.2 Reconstruction et suivi de surfaces

En infographie, ce qui nous intéresse au final est de pouvoir *visualiser* le résultat de notre simulation, soit la surface dans le cas d'une simulation de liquide. Nous verrons donc dans cette section différentes techniques représentant l'état de l'art en reconstruction de surface de liquide basé sur différents types de simulations.

2.2.1 Surfaces implicites

On dit qu'une surface est implicite lorsque sa représentation nous permet de connaitre tous les points appartenant à celle-ci, mais ce, sans nous donner directement l'ensemble de ces points (auquel cas elle serait *explicite*). L'un des premiers travaux de reconstruction de surfaces implicites fut réalisé par Blinn [Bli82] qui a introduit le concept de molécule "blobby". Cette technique permet la modélisation de surfaces implicites complexes à partir d'une somme pondérée de fonctions "noyaux" décroissantes plus simples. Cette méthode fut d'abord inventée afin de permettre la visualisation de molécules, mais peut également s'appliquer à la visualisation d'objets souples, tel qu'illustré à la figure 2.4, à droite.

Une des applications du modèle de Blinn fut utilisée dans le domaine de la simulation de fluides par Müller et al. [MCG03] afin non seulement de reconstruire la surface d'un ensemble de particules, mais également d'approximer la courbure à la surface d'une simulation SPH et de permettre de modéliser la tension de surface sur les particules. Dans ce modèle, chaque particule contribue à la fonction implicite à $\mathbf{x}$ en fonction de sa distance à $\mathbf{x}$ (plus de détails à la section 3.2 et illustré à la figure 3.1). Cela a l'avantage d'être très simple et efficace, mais donne une apparence "blobby" à la surface, rendant évidente la nature lagrangienne de la simulation sous-jacente. En effet, à la figure 2.5 (en haut à gauche), il est très aisé de discerner les particules sphériques étant donnée l'échelle macroscopique de la simulation combinée à l'usage des supports isotropes de la méthode de Müller et al.

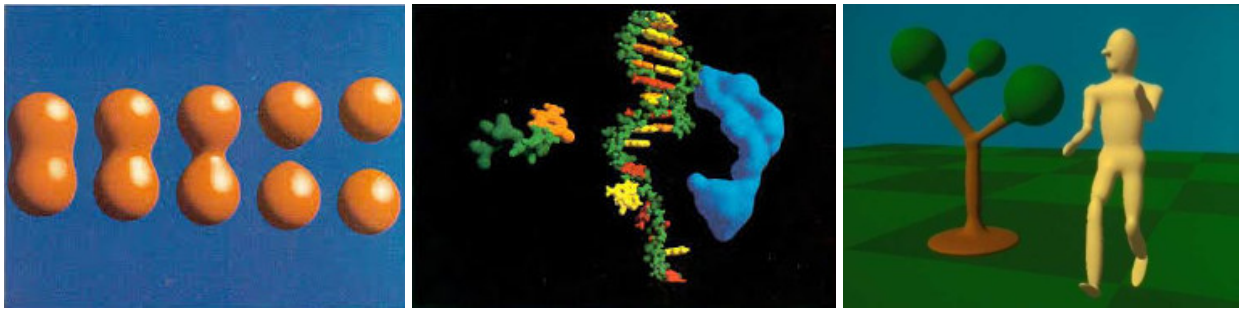


Figure 2.4 – Molécule “blobby” de Blinn [Bli82]. À gauche, les différences entre les surfaces implicites obtenues avec deux noyaux de fonctions isotropes lorsque la distance entre ces noyaux varie. Au centre, le modèle de Blinn est appliqué afin de permettre la visualisation de molécules. À droite, des objets souples sont obtenus grâce à l’usage d’une grande variété de noyaux d’influence isotropes (par exemple pour le feuillage) et anisotropes (par exemple pour les branches, la base de l’arbre et les membres du personnage, etc.).

Partant du principe que pour une seule particule, la surface devrait se situer sur son rayon, Zhu et Bridson [ZB05] ont proposé une nouvelle fonction implicite où la distance à la surface à $\mathbf{x}$ est plutôt évaluée comme la distance à la surface d’une particule “moyenne” dans le voisinage de $\mathbf{x}$. Cette méthode permet efficacement d’éliminer l’apparence “blobby” de la surface puisque cette formulation simule en quelque sorte l’existence de particules *entre* les particules de la simulation. Par contre, il peut également arriver qu’une “particule moyenne” se retrouve erronément *à l’extérieur* du fluide réel. Cela crée des bosses ou des pointes sur la surface et rend impératif l’emploi d’une seconde passe de lissage sur la surface générée afin de réduire ces artefacts. Cela est non seulement coûteux en temps de calcul, mais également en espace puisqu’il est nécessaire de stocker les valeurs de distances à la surface dans une grille 3D haute résolution afin de procéder au lissage. De plus, le modèle de Zhu et Bridson se base sur la prémisse que la distance à la surface d’une particule à la surface est égale à son rayon, ce qui n’est pas nécessairement vrai. Une légère modification a été appliquée par Adams et al. [APKG07] à la méthode de Zhu et Bridson afin de prendre ce dernier problème en compte. La méthode d’Adams et al. évalue la distance des particules à la surface au cours du temps pendant la simulation et propage ces distances des particules de surfaces vers les particules d’intérieur par une méthode de *Fast Marching* [Set95]. La formule implicite de Zhu et Bridson est ensuite utilisée en remplaçant les rayons des particules par les distances à la surface calculées. La méthode parvient à

obtenir des surfaces plus lisses, mais hérite des artefacts de bosses et de pointes à la surface pour les mêmes raisons.

Tous ces modèles ont un problème en commun : ils n'approximent pas bien les surfaces planes. Dans le but de résoudre ce problème, Yu et Turk [YT10] ont proposé un modèle afin d'obtenir des surfaces plus lisses. Leur modèle est similaire à celui de Müller et al. à la différence qu'il utilise des supports *anisotropes* à chaque particule, c'est-à-dire un support qui varie en fonction de la direction selon laquelle on l'évalue. Plus spécifiquement, ils utilisent des supports ellipsoïdaux plutôt que sphériques. À chaque pas de simulation, ils font une analyse pondérée des principaux composants (*Weighted Principal Component Analysis* (WPCA)) à chaque particule afin d'en déterminer l'anisotropie. Ainsi les particules par exemple à la surface auront un support "aplati" et la surface obtenue en sera donc d'apparence plus plane. Leur modèle donne donc de meilleurs résultats visuels au coût de devoir résoudre un très grand nombre de WPCA.

2.2.2 Surfaces explicites

Lorsqu'arrive le temps de rendre une image de la surface du fluide, la plupart des moteurs de rendu ont besoin d'un modèle *explicite* de la surface du fluide, c'est-a-dire un format qui donne *directement* tous les points appartenant à la surface plutôt qu'une formule permettant de retrouver la surface. Les modèles explicites les plus utilisés sont les maillages polygonaux. Il est bien sûr possible d'extraire une surface explicite étant donné un modèle implicite par des méthodes telles que le *Marching Cubes* [LC87]. C'est d'ailleurs précisément ce type d'algorithme qui a été utilisé pour créer les maillages des surfaces implicites de la figure 2.5. Par contre, les algorithmes de ce type ont leurs limites. Ils peuvent perdre des éléments fins de la simulation lorsque ceux-ci se trouvent entièrement à l'intérieur d'un *marching cube*. Aussi, si on reconstruit une surface explicite par *Marching Cubes* à chaque image, on n'a absolument aucune assurance d'obtenir une cohérence temporelle et spatiale entre les sommets de deux maillages consécutifs. Ceci pose problème lorsqu'on souhaite transporter des quantités sur la surface du maillage (des textures par exemple).

Un autre modèle de localisation de surface strictement explicite a été proposé par Williams [Wil08]. Dans cet algorithme, chaque particule est traitée comme une sphère avec deux rayons : un rayon interne

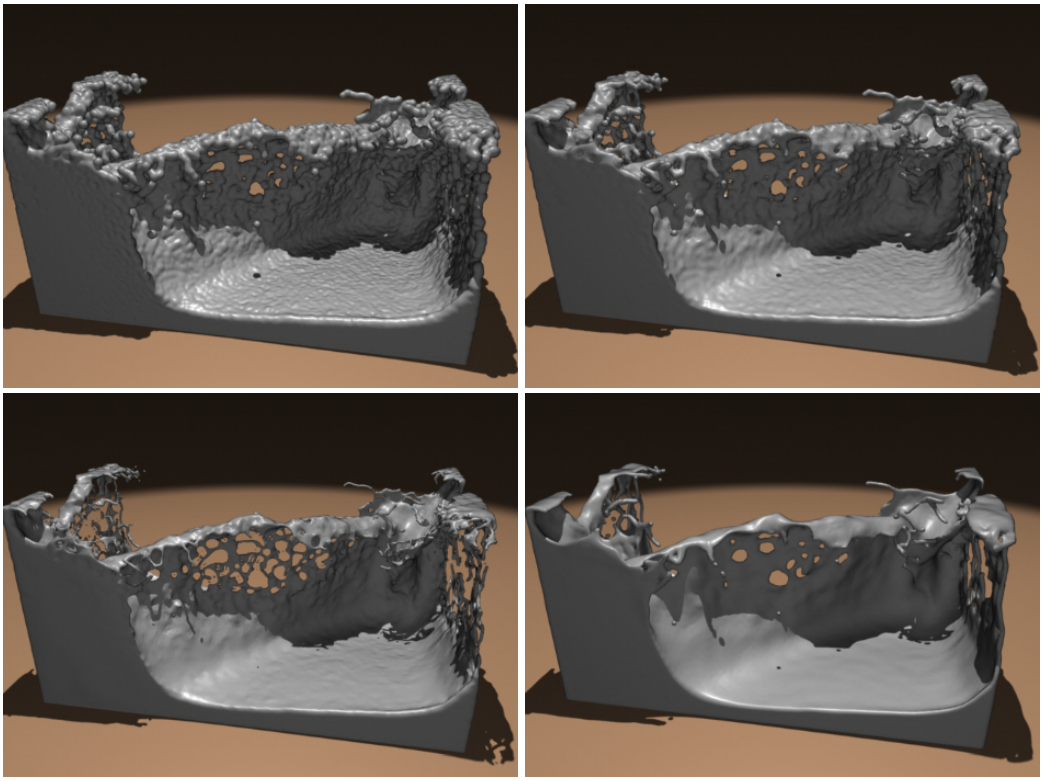


Figure 2.5 – Différentes techniques afin d'extraire la surface d'une simulation de fluide basée sur un ensemble de particules. En haut à gauche, le modèle de Müller et al. [MCG03] utilise des noyaux isotropes ce qui rend la nature lagrangienne de la simulation évidente. En haut à droite, Zhu et Bridson [ZB05] utilisent des valeurs de distances à une particule "moyenne" dans un voisinage et font une passe de lissage pour éliminer les artéfacts occasionnés par l'ajout erroné de particules "moyennes". En bas à gauche, Adams et al. [APKG07] modifient le modèle de Zhu et Bridson en évaluant les distances des particules à la surface au cours du temps et en utilisant ces distances variables plutôt que les rayons des particules. En bas à droite, Yu et Turk [YT10] utilisent des noyaux anisotropes pour chaque particule dont l'influence et l'orientation varient en fonction de la distribution des particules dans un voisinage.

et un rayon externe. Ces rayons permettent de délimiter une zone dans laquelle la surface devrait se situer. L'algorithme commence par créer un maillage sur les rayons *externes* à l'aide d'un algorithme de *Marching Tiles* (sa propre variante de *Marching Cubes*), puis il lisse itérativement la surface par minimisation de l'énergie de celle-ci donnée par l'opérateur bilaplacien sous les contraintes de devoir demeurer entre les rayons internes et externes. À la figure 2.6, à gauche, on voit un exemple simple d'une surface obtenue. Les points gris indiquent la position des particules avec leur centre. Le maillage est initialisé sur la ligne grise et converge vers la ligne orange après quelques itérations de leur algorithme. À droite on peut voir quelques-uns des résultats obtenus avec cette méthode. Cette méthode a l'avantage de créer une surface particulièrement fidèle à la simulation sous-jacente tout en n'ayant pas les artefacts d'apparence "blobby" présents dans un grand nombre de méthodes utilisant un modèle implicite. Par contre, cette méthode peut également souffrir de problèmes de cohérence temporelle. Puisqu'à chaque image, la surface est réinitialisée et réitérée afin de la rendre lisse, il est possible que pour deux images consécutives elle ne converge pas vers deux surfaces consécutives cohérentes entre elles. Ceci est d'autant plus probable puisque l'algorithme de *Marching Tiles* ne garantit en rien que les sommets auront les mêmes voisins ni la même valence.

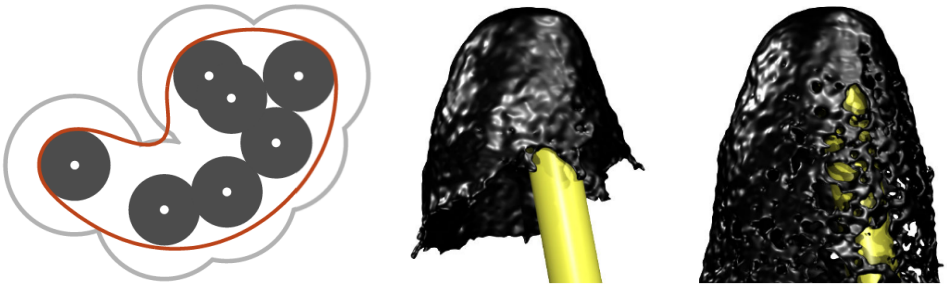


Figure 2.6 – Reconstruction de surface avec la méthode de Williams [Wil08]. À gauche : illustration en 2D d'une surface obtenue avec cette méthode (ligne orange). À droite : quelques résultats obtenus.

2.2.3 Suivi de surfaces

Il est souvent souhaité en infographie que deux maillages consécutifs d'une animation aient la même connectivité entre les sommets, c'est-à-dire que ce soit le *même* maillage avec translation des

sommets. Ceci permet entre autres de transporter aisément des quantités sur la surface telles que des coordonnées de texture, des couleurs, des températures, des valeurs de dynamique de surface, etc. Cependant, il n'est généralement pas possible d'assurer cette relation entre deux surfaces consécutives de fluide en pratique à cause des nombreux changements de topologie. Par contre, on peut tout de même tenter de maximiser la bijection entre les sommets de ces maillages de telle sorte que pour deux maillages M_n et M_{n+1}, M_{n+1} soit la *transformation* de M_n obtenue en fonction des mouvements du fluide (illustré dans la figure 2.7).

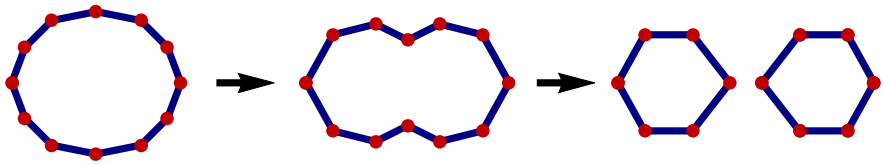


Figure 2.7 – Illustration de changement de topologie sur un maillage 2D simple. Le maillage d'une gouttelette M_n (à gauche) est transformé en un maillage de même topologie M_{n+1} (au centre), puis la gouttelette se sépare en deux et il n'est plus possible d'obtenir une bijection entre les sommets du nouveau maillage M_{n+2} (à droite) avec les deux précédents.

Plusieurs travaux ont été réalisés dans cette direction pour des simulations eulériennes, par exemple les travaux de Wojtan [WTGT09, BHW13] et Brochu [BBB10], mais très peu pour les simulations lagrangiennes. C'est ce qu'ont tenté de résoudre Yu et al. [YWTY12] en utilisant une méthode semi-explicite. Leur technique initialise d'abord un maillage par une méthode de *Marching Cubes* sur une surface implicite définie par la méthode de Yu et Turk [YT10]. Ce maillage est ensuite advecté avec la simulation de fluide, optimisé, puis projeté sur la surface implicite du nouvel état de la simulation. Lorsque l'étape de projection échoue pour certains sommets, c'est-à-dire que la surface implicite n'a pas été traversée en faisant un pas dans la direction du gradient, cela indique qu'il y a eu un changement de topologie et le maillage est donc reconstruit à ces endroits. Le problème de cette technique est qu'il lui arrive de perdre la trace de certains éléments de la simulation. Spécifiquement, il peut arriver que l'étape de projection des sommets sur la surface implicite ne détecte pas correctement les particules isolées qui atterrissent donc *à l'extérieur* de la surface. Cela donne des résultats visuels très étranges non seulement parce que des structures disparaissent, mais également parce que lorsque ces structures "invisibles" rejoignent les visibles, elles modifient évidemment la surface implicite et créent donc un

trou ou une bosse à la surface de manière complètement inattendue.

2.3 Modèles de turbulence

Puisque les fluides non visqueux ont généralement un haut contenu en fréquences, les résultats des simulations sont généralement insuffisants et l'on a recours à des modèles de turbulences afin de réinjecter les détails perdus par la simulation initiale. Certains de ces modèles s'appliquent pendant la simulation alors que d'autres sont découplés de la simulation et s'appliquent après. Nous verrons dans cette section un aperçu de certaines de ces techniques.

Une méthode largement utilisée en simulation de fumée est celle du *vorticity confinement* [JS94] dû à sa simplicité et son efficacité. Cette méthode s'utilise pendant la simulation et consiste à détecter les endroits où se forment les tourbillons, puis à injecter des forces afin de renforcer ceux-ci. Il suffit de calculer le rotationnel du champ vectoriel des vélocités de la simulation afin d'avoir une mesure de la rotation. La direction du rotationnel correspond à l'axe de rotation et la magnitude correspond à la magnitude de la rotation. En faisant le produit croisé du rotationnel normalisé avec le gradient de la magnitude du rotationnel, on obtient donc un vecteur pointant dans la direction du tourbillon qui peut ensuite être utilisé comme un vecteur de force à appliquer à la simulation. La figure 2.8 illustre un résultat obtenu par cette méthode ainsi qu'une visualisation du rotationnel en 2D. Malheureusement, bien que cette méthode permette de réduire considérablement les artéfacts de dissipation numérique, elle ne permet pas d'augmenter la résolution de la simulation puisqu'elle agit sur la même grille (de même résolution).

Une autre méthode qui a également reçu beaucoup d'attention est celle du *wavelet turbulence* [KTJG08] pour les simulations de fumée. Cette méthode s'applique totalement en tant que post-processus sur des simulations eulériennes et/ou lagrangiennes. Elle consiste à faire une décomposition en ondelettes (*wavelet decomposition*) afin de détecter les endroits sous-résolus et d'appliquer une fonction de turbulence incompressible basée sur la théorie de turbulence de Kolmogorov afin de réintroduire ces détails arbitrairement. Cette méthode est d'autant plus intéressante qu'elle ne nécessite aucune résolution de système linéaire et se parallélise donc aisément. La figure 2.9 montre un exemple

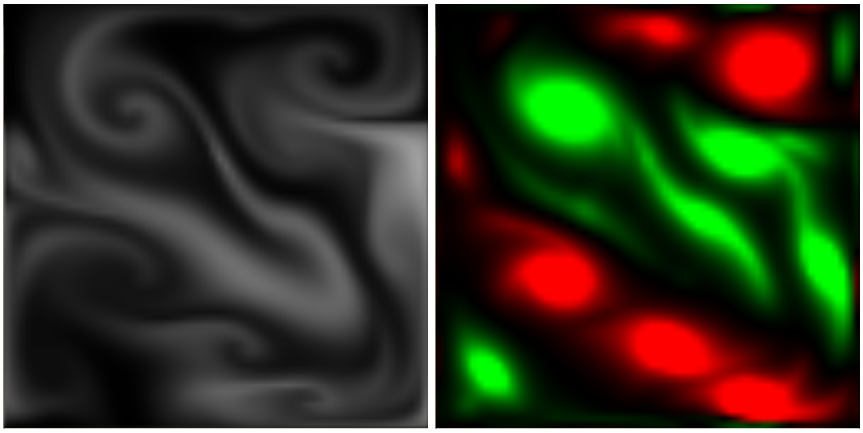


Figure 2.8 – Résultats obtenus avec la méthode du *vorticity confinement*. À gauche, une simulation de fumée en 2D dans une boîte où l'on peut voir les tourbillons renforcés par la méthode. À droite, une visualisation de la magnitude du rotationnel du champ vectoriel des vélocités de la simulation utilisé afin de réinjecter des forces dans le système (en vert pour les valeurs positives, en rouge pour les valeurs négatives).

de résultat obtenu avec cette méthode.

Ces méthodes donnent de bons résultats pour des simulations de fumée, mais ne permettent pas d'augmenter la résolution apparente sur la surface de simulations de liquides. Les méthodes d'augmentation de résolution du champ de vélocité donnent d'ailleurs de bien piètres résultats puisque la surface du liquide possède sa propre dynamique (tension de surface, vagues capillaires, etc.) et est très faiblement influencée par le champ de vélocité. C'est ce que tente de résoudre la méthode du *Closest Point Turbulence* [KTT13] illustré à la figure 2.10. Cette méthode s'applique en tant que post-processus sur une simulation eulérienne. Elle construit une bande d'extension autour du *levelset* nul (c.-à-d. de la surface) de la simulation d'entrée dans laquelle elle simule l'algorithme iWave [TCT99]. Elle injecte de la turbulence aux endroits où la courbure du fluide est près de la limite de Nyquist puis utilise la méthode du point le plus près (*Closest Point Method*) [RM08] afin de résoudre les équations différentielles partielles à la surface et propager les vagues injectées. Malheureusement, cette méthode ne se généralise pas pour les simulations lagrangiennes puisqu'elle crée la bande d'extension à partir de la représentation eulérienne de la simulation d'entrée.

L'une des premières méthodes à avoir ajouté des détails à la surface de simulations lagrangiennes

Figure 2.9 – Comparaisons des résultats de simulation sans et avec *wavelet turbulence* [KTJG08]. À droite, une fonction de turbulence incompressible basée sur la théorie de Kolmogorov est appliquée sur la simulation d'entrée suréchantillonnée (à gauche) afin d'injecter des détails haute fréquence.

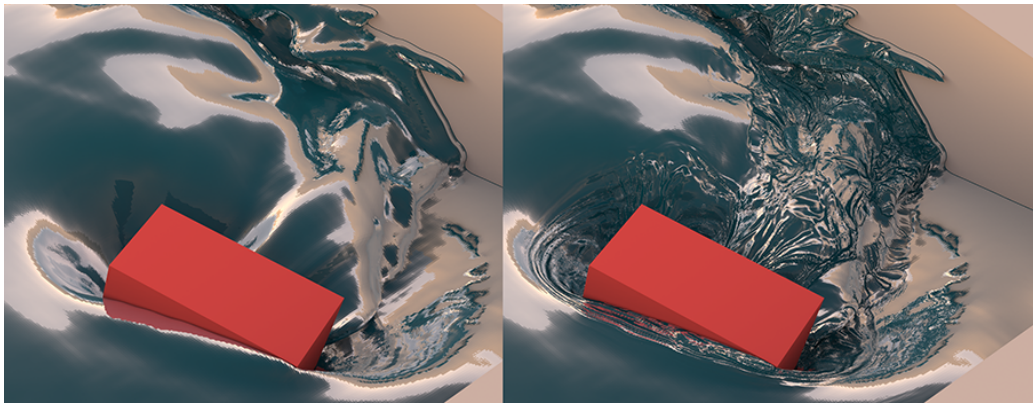


Figure 2.10 – Résultats de la méthode *Closest Point Turbulence* [KTT13]. L'algorithme de iWave 3D est résolu dans une bande d'extension autour de la surface d'entrée (à gauche) permettant d'augmenter considérablement la résolution apparente de celle-ci (à droite).

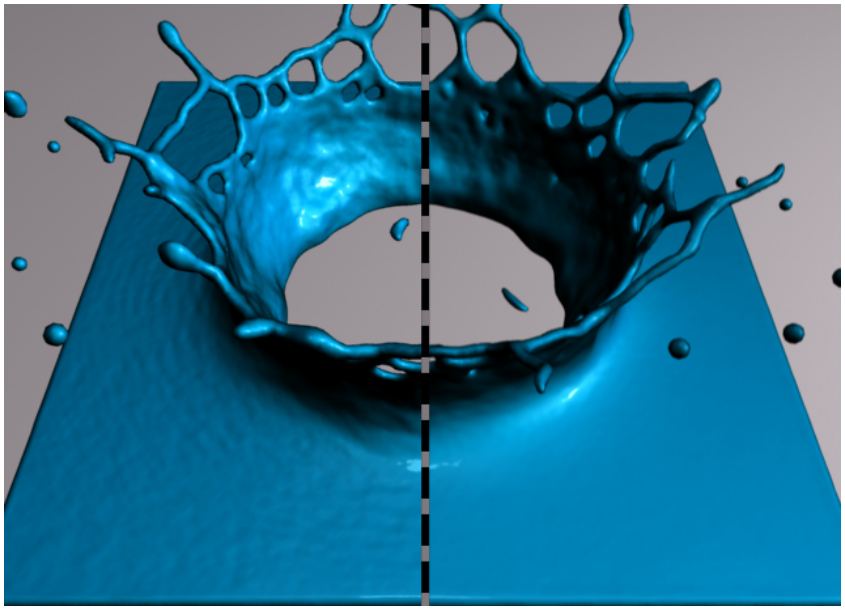


Figure 2.11 – Modèle de turbulence de Yu et al. [YWTY12]. La dynamique de surface est ajoutée par minimisation de l'énergie par rapport au plan afin d'ajouter de la turbulence sur la surface (à gauche) étant donné le maillage initial (à droite).

(plus spécifiquement SPH) est celle de Yu et al. [YWTY12] (figure 2.11). Des valeurs de vagues (hauteur et vélocité) sont ajoutées et transportées directement sur le maillage de la surface. Les vagues capillaires sont simulées par minimisation de l'énergie de la surface par rapport au plan en plus d'une contrainte sur l'énergie entre la surface et sa déformation afin d'éviter que la surface déplacée ne perde trop de volume ou ne dévie trop par rapport à la surface non déplacée. Puisque cette méthode s'applique sur les sommets d'un maillage, il est impératif que celui-ci soit bien triangulé et que les triangles soient non dégénérés afin d'éviter les instabilités numériques. De plus, leur modèle interdit l'ajout de turbulences aux endroits où la courbure est trop élevée car cela résulte en une dynamique de surface bruitée. Notre méthode utilise au contraire ces régions afin d'injecter de la turbulence puisque c'est précisément dans ces régions que les hautes fréquences sont perdues et évite le problème des triangles dégénérés en utilisant un modèle sans maillage.

De manière totalement orthogonale, certains travaux augmentent le réalisme des simulations non pas en réinjectant des hautes fréquences, mais plutôt en ajoutant les détails dus aux interactions eau-air

notables dans les fluides turbulents, à savoir les éclaboussures, l'écume ainsi que les bulles d'airs capturées sous l'eau, tel qu'illustré à la figure 2.12. C'est notamment ce que résolvent Ihmsen et al. [IAAT12] grâce à différentes heuristiques basées sur la courbure et la vélocité afin de déterminer où ajouter ces éléments diffus. Nous considérons ce champ de recherche comme étant orthogonal et complémentaire au nôtre. Ces méthodes peuvent s'appliquer conjointement avec les autres méthodes mentionnées ci-haut et tel que démontré dans notre article.

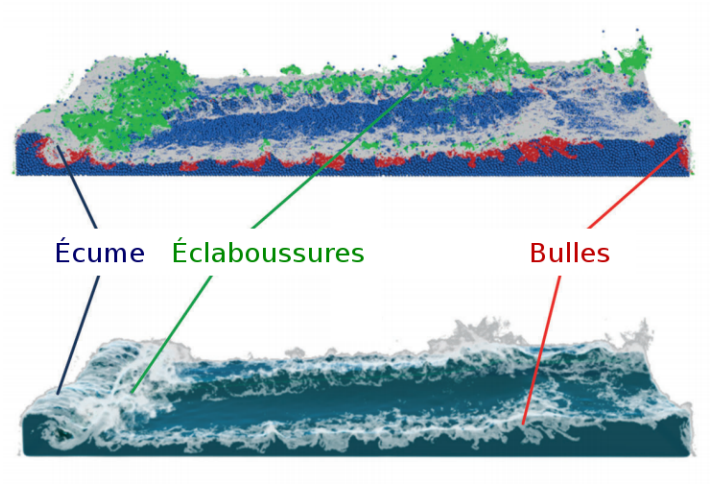


Figure 2.12 – Méthode d'augmentation de réalisme par ajout d'effets d'éclaboussures, de bulles et d'écume. [IAAT12]

Chapitre 3

Contexte Technique

3.1 Simulations FLIP

Tel qu'introduit dans la section précédente, les simulations FLIP combinent les approches eulériennes et lagrangiennes afin de résoudre le continuum des fluides. Soient les équations de Navier-Stokes décrivant le mouvement des fluides :

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f} \tag{3.1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{3.2}$$

où $\mathbf{u}$ est la vélocité, ρ la densité, p la pression, ν la viscosité, $\mathbf{f}$ un vecteur de forces externes, ∇ l'opérateur gradient, $\nabla\cdot$ l'opérateur de divergence et ∇^2 est l'opérateur laplacien. Plus précisément, soit f un scalaire ou un vecteur dans $\mathbb{R}^3$, et $\mathbf{u} = (u,v,w)^T$, alors les opérateurs sont définis par,

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}, \qquad \nabla\cdot\mathbf{u} = \frac{\partial \mathbf{u}}{\partial x} + \frac{\partial \mathbf{u}}{\partial y} + \frac{\partial \mathbf{u}}{\partial z} \qquad \text{et} \qquad \nabla\times\mathbf{u} = \begin{pmatrix} \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} \\ \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \end{pmatrix}.$$

L'équation 3.1 décrit comment le mouvement du fluide est influencé par les forces internes (vélocité $-(\mathbf{u}\cdot\nabla)\mathbf{u}$, pression $-\frac{1}{\rho}\nabla p$, viscosité $\nu\nabla^2\mathbf{u}$) et externes (obstacles, vent, etc. ; $\mathbf{f}$) par rapport au temps t. L'équation 3.2 est une contrainte servant à garantir la non-divergence du champ de vélocité et ainsi assurer la conservation de la masse. Intuitivement, un champ vectoriel est non divergent s'il est incompressible, c'est-à-dire si, pour tout volume V de fluide à l'intérieur de ce champ, le flot entrant aux frontières de ce volume est égal au flot sortant. Le contraire impliquerait que le fluide se compresserait ou se décompresserait. Tenter de résoudre les équations de Navier-Stokes en un seul bloc résulte en un système non linéaire difficile à résoudre. Afin de simplifier le problème, on fragmente l'équation en sous-problèmes simplifiés. Plus spécifiquement, les différents termes de l'équation sont résolus séquentiellement en utilisant l'approche la mieux adaptée pour résoudre le problème.

3.1.1 Algorithme

Marquage des cellules pleines. Afin de résoudre l'ajout des forces et la pression, il est nécessaire de différencier les cases contenant le liquide des cases contenant de l'air. En effet, lors de la résolution de la pression, il est courant de faire l'assomption que l'air exerce une pression nulle sur le liquide et que celui-ci peut se déplacer librement dans les cases vides. Cette assomption est fausse mais est une approximation raisonnable. Cette étape consiste donc simplement à marquer comme "pleine" toute case de la grille contenant au moins une particule, "vide" une case contenant de l'air et "solide" une case contenant un obstacle.

Transfert des vélocités des particules vers la grille. Dans le modèle FLIP, ce sont les particules qui définissent le fluide et transportent ses quantités (température, vélocité, etc.) mais c'est sur la grille que la majorité des calculs sont faits. Ainsi, on doit transférer les quantités nécessaires des particules vers la grille à chaque itération par moyenne pondérée des particules avoisinantes (équation 3.3). Les quantités sont stockées au centre des cases de la grille et les vélocités sont stockées sur les frontières de celles-ci en utilisant une grille MAC (c'est pourquoi on peut voir des "demi-indices" dans l'équation suivante. Nous référons le lecteur aux travaux de Harlow et Welch [HW65] pour plus de détail sur les grilles MAC),

$$u_{i+1/2,j,k} = \frac{\sum_p u_p k(\mathbf{x}_p - \mathbf{x}_{i+1/2,j,k})}{\sum_p k(\mathbf{x}_p - \mathbf{x}_{i+1/2,j,k})} \tag{3.3}$$

où k est un noyau de pondération (généralement bilinéaire en 2D ou trilinéaire en 3D).

Résolution de tous les termes à l'exception de celui de l'advection. C'est à cette étape que sont résolus l'ajout des forces externes, le calcul de la dissipation due à la viscosité, ainsi que l'ajout des forces occasionnées par la pression interne. En pratique, puisque l'un des buts principaux de FLIP est d'éviter la dissipation numérique rendant le fluide erronément visqueux, on simule souvent des fluides dont la viscosité cinétique ν est presque nulle, et il donc très courant de négliger complètement ce terme. On tente plutôt de résoudre l'équation non visqueuse d'Euler,

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u} - \frac{1}{\rho} \nabla p + \mathbf{f} \,.$$

On a donc deux termes à résoudre sur la grille : l'ajout des forces externes et l'ajout des forces dues à la pression interne. L'ajout des forces externes est trivial : on ajoute simplement l'accélération multipliée par le pas de temps au champ des vélocités. On obtient un nouveau champ de vélocité divergent qui ne respecte pas la loi de conservation de la masse et on doit donc effectuer une projection afin d'y remédier. Pour ce faire, on doit résoudre une équation de Poisson. Cette équation est obtenue à partir de la décomposition d'Helmholtz-Hodge qui dit que tout champ vectoriel suffisamment lisse peut être représenté par la somme d'un champ vectoriel non divergent et d'un champ vectoriel non rotationnel. Ceci implique que tout champ vectoriel satisfaisant cette condition peut être exprimé par une somme de deux potentiels : un potentiel scalaire et un potentiel vectoriel. En effet, soit $\mathbf{w}$ le champ de vélocité courant (divergent), la décomposition Helmholtz-Hodge nous donne,

$$\mathbf{w} = \mathbf{u} + \nabla p \quad \Rightarrow \quad \mathbf{u} = \mathbf{w} - \nabla p. \tag{3.4}$$

où $\mathbf{u}$ est un champ vectoriel non divergent et ∇p est un champ vectoriel non rotationnel. Un champ vectoriel est non rotationnel si, pour tout point dans ce champ, la rotation locale en ce point est nulle. La rotation locale est mesurée par l'opérateur rotationnel et on obtient effectivement que ∇p est non rotationnel puisque $\nabla \times \nabla p = \mathbf{0}$, tel que le montre le développement suivant :

$$\nabla \times \nabla p = \nabla \times \begin{pmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \\ \frac{\partial p}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{\partial^2 p}{\partial y \partial z} - \frac{\partial^2 p}{\partial z \partial y} \\ \frac{\partial^2 p}{\partial z \partial x} - \frac{\partial^2 p}{\partial x \partial z} \\ \frac{\partial^2 p}{\partial x \partial y} - \frac{\partial^2 p}{\partial y \partial x} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.5}$$

En appliquant l'opérateur de divergence de chaque côté, on obtient

$$\nabla \cdot \mathbf{w} = \nabla \cdot \mathbf{u} + \nabla \cdot \nabla p = \nabla \cdot \nabla p. \tag{3.6}$$

On obtient donc l'équation de Poisson à résoudre afin de connaître le champ de pression, ce qui peut être fait à l'aide de différentes méthodes telles que Gauss-Seidel, Jacobi ou gradient conjugué. Une fois p résolu, on soustrait son gradient au champ de vélocité du fluide courant afin de le rendre non

divergent à nouveau, c'est-à-dire, assurer que $\nabla \cdot \mathbf{u} = 0$ est respecté.

Transfert des vélocités de la grille vers les particules. Contrairement à la méthode PIC [Har63] qui consiste à *écraser* les vélocités des particules avec l'interpolation de la vélocité dans la grille à sa nouvelle position, la méthode FLIP vient plutôt *modifier* ces dernières par la différence entre les anciennes valeurs dans la grille et les nouvelles. Cela permet de réduire le lissage entraîné par l'interpolation dans la méthode PIC. En pratique, on utilise souvent une combinaison linéaire de PIC et FLIP de la forme :

$$\mathbf{u}_p^{\text{new}} = (1-\alpha)\text{interp}(\mathbf{u}_{\text{grid}}^{\text{new}}, \mathbf{x}_p) + \alpha \left[\mathbf{u}_p^{\text{old}} + \text{interp}(\Delta \mathbf{u}_{\text{grid}}, \mathbf{x}_p)\right]. \tag{3.7}$$

On obtient un pur transfert PIC lorsque $\alpha = 0$ et un pur transfert FLIP lorsque $\alpha = 1$. Une analyse de la dissipation numérique entraînée par PIC a montré que α peut être associé à la viscosité cinétique ν par la formule :

$$\alpha = 1 - \frac{6\Delta t \nu}{\Delta x^2} \tag{3.8}$$

On peut donc arbitrairement modifier α afin d'obtenir une certaine viscosité désirée. Bien sûr, si $\alpha < 0$, il doit être tronqué à 0 et il faudra résoudre l'étape de diffusion ignorée dans l'équation non visqueuse d'Euler pour obtenir une plus grande viscosité.

Advecter les particules. Finalement, on advecte les particules avec les vélocités de la grille par une méthode d'intégration numérique, par exemple une méthode d'Euler ou encore Runge-Kutta de second ou plus grand degré.

3.1.2 Observations

Bien que la grille eulérienne soit utilisée pour la majorité des calculs, elle ne définit pas le fluide ; ce sont les particules qui le définissent. Ce sont elles qui déterminent où le fluide se trouve et chaque case de la grille eulérienne contient plusieurs particules. Utiliser la grille eulérienne afin de reconstruire la surface résulterait donc en une perte d'information et on optera plutôt pour des méthodes de reconstruction utilisant les particules.

Ce qu'on observe également du modèle FLIP est que, contrairement au modèle SPH, la distribution

des particules est hautement non uniforme. Il est donc impératif de pondérer l'influence des particules par l'inverse de leur densité pour reconstruire la surface. Autrement les régions de haute densité de particules gagneraient du volume, ce qui serait erroné puisque les particules FLIP ne sont que des échantillons permettant de stocker des valeurs du fluide et marquer les cases pleines mais n'ont pas de volume en soi. La densité des particules n'influence aucunement les calculs de résolution de la pression.

3.2 Surface

Afin de faire le rendu d'une simulation de liquide ou encore d'ajouter des détails de dynamiques de surface, il nous faut d'abord reconstruire la surface de celle-ci. Nous verrons dans cette section différentes représentations et méthodes applicables à des simulations basées sur un système de particules afin d'y parvenir.

3.2.1 Surfaces implicites

Il y a deux techniques principales utilisées afin de mathématiquement définir une surface dans l'espace Euclidien $\mathbb{R}^3$: paramétrique et implicite.

Les surfaces paramétriques sont définies par une équation paramétrique avec deux paramètres de surface (s,t) et sont de la forme :

$$\mathbf{x}(s,t) = (x(s,t), y(s,t), z(s,t)) \ . \tag{3.9}$$

Ce type de formulation est largement utilisé afin simuler des eaux peu profondes ou des océans en faisant l'assomption que ceux-ci sont suffisamment calmes pour être représentés par une carte de hauteurs (*height field*). Par contre, lorsqu'on travaille avec des simulations plus complexes avec des éclaboussures et des fluides qui se séparent et fusionnent, ce modèle devient inadapté et les surfaces implicites sont alors préférables.

Une surface implicite correspond au *level set* nul d'une fonction à trois variables. Un *level set* est l'ensemble des points d'une fonction telle que la fonction évaluée en ce point donne une même

constante c (le *level*). Les surfaces implicites S sont donc définies par l'ensemble de points $\mathbf{x} \in \mathbb{R}^3$ satisfaisants une équation $\phi(\mathbf{x}) = 0$, c'est-à-dire

$$S = \{\mathbf{x} \,:\, \phi(\mathbf{x}) = 0\} \,. \tag{3.10}$$

La surface implicite est donc l'ensemble des zéros d'une fonction à trois variables. Cette formulation est largement utilisée en infographie, ne serait-ce que pour représenter des formes primitives simples telles que des coniques. Par exemple :

$$x^2 + y^2 - R^2 = 0 \quad \text{(cylindre)} \tag{3.11}$$

$$x^2 + y^2 + z^2 - R^2 = 0 \quad \text{(sphère)} \tag{3.12}$$

$$(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(x^2 + y^2)^2 = 0 \quad \text{(torus)} \tag{3.13}$$

Les surfaces illustrées à la figure 2.5 sont toutes créées à partir de ce type de formulation, mais avec différentes définitions de fonction pour $\phi(\mathbf{x})$. Comme indiqué dans la section 2.2, les travaux de Blinn [Bli82] furent pionniers dans le domaine et demeurent la base de nombreuses techniques de localisation de surface de fluides lagrangiens. Il a introduit le concept de *noyau* (*kernel*), une fonction scalaire décrivant l'influence d'un élément (dans notre contexte d'une particule) dans son voisinage. Un exemple de noyau pour une particule i pourrait être

$$\phi_i(\mathbf{x}) = w_i \exp(-a \, ||\mathbf{x} - \mathbf{x}_i||^2) \tag{3.14}$$

avec w_i un poids accordé à cette particule (par exemple l'inverse de sa densité). Cette équation correspond au noyau illustré à la figure 3.1 (en haut à gauche). La fonction implicite correspond à la somme des noyaux et la *surface* implicite correspond à une coupe dans cette fonction.

$$\phi(\mathbf{x}) = \sum_i \phi_i(\mathbf{x}) \tag{3.15}$$

Ce type de surfaces implicites étant décrit par une somme de fonctions, il est donc très facile

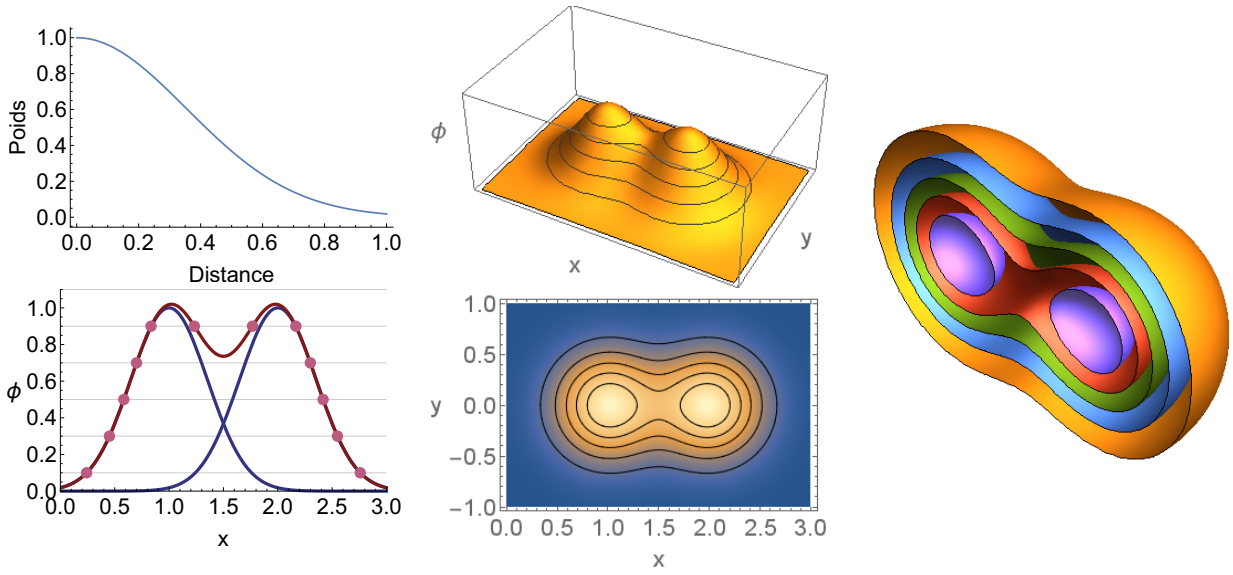


Figure 3.1 – Noyaux d'influence en reconstruction de surface. En haut à gauche, un exemple de noyau $f(d) = \exp(-4d^2)$. De gauche (en bas) à droite, $\phi(\mathbf{x}) = \sum_p f(|\mathbf{x} - \mathbf{p}_x|)$ avec $\mathbf{x} \in \mathbb{R}$, $\mathbb{R}^2$, et $\mathbb{R}^3$ respectivement. Un *levelset* correspond à une coupe dans la fonction, c'est-à-dire l'ensemble des points satisfaisant $\phi(\mathbf{x}) = c$ pour une constante c.

d'obtenir des informations sur celles-ci. Par exemple, soit $\mathbf{x}$ un point appartenant à la surface, alors la normale de la surface à $\mathbf{x}$ est obtenue par

$$\mathbf{n}(\mathbf{x}) = \frac{-\nabla\phi(\mathbf{x})}{|\nabla\phi(\mathbf{x})|} \tag{3.16}$$

ce qui est généralement très facile à résoudre puisque $\phi(\mathbf{x})$ est une somme de fonctions simples et donc la dérivée de $\phi(\mathbf{x})$ est la somme des dérivées de ces fonctions simples. La courbure gaussienne K_G et la courbure moyenne K_M peuvent également être obtenues par les équations suivantes

$$K_G(\mathbf{x}) = \frac{\nabla\phi(\mathbf{x})^T \cdot H^*(\phi(\mathbf{x})) \cdot \nabla\phi(\mathbf{x})}{|\nabla\phi(\mathbf{x})|^4}, \tag{3.17}$$

$$K_M(\mathbf{x}) = \frac{\nabla\phi(\mathbf{x})^T \cdot H(\phi(\mathbf{x})) \cdot \nabla\phi(\mathbf{x}) - |\nabla\phi(\mathbf{x})|^2 \mathrm{Trace}(H(\phi(\mathbf{x})))}{2|\nabla\phi(\mathbf{x})|^3}, \tag{3.18}$$

où H est la matrice hessienne. Par contre, dans notre article, la surface reconstruite n'est pas une surface implicite, mais utilise plutôt une fonction implicite afin de contraindre la surface reconstruite à

demeurer proche des particules tout en étant lisse. Plus spécifiquement, nous construisons une surface par point où chaque point doit demeurer entre le niveau 0 et le niveau 1 d'une fonction implicite (plus de détails au chapitre 4). Par conséquent, puisque chaque point de surface existe à un niveau $\phi_i \in [0,1]$ différent du niveau $\phi_j \in [0,1]$ de son point voisin, alors l'évaluation du gradient, de la courbure moyenne et de la courbure gaussienne utilisant la fonction contrainte ϕ est incohérente par rapport à la vraie surface par points obtenue, tel qu'illustré à la figure 3.3.

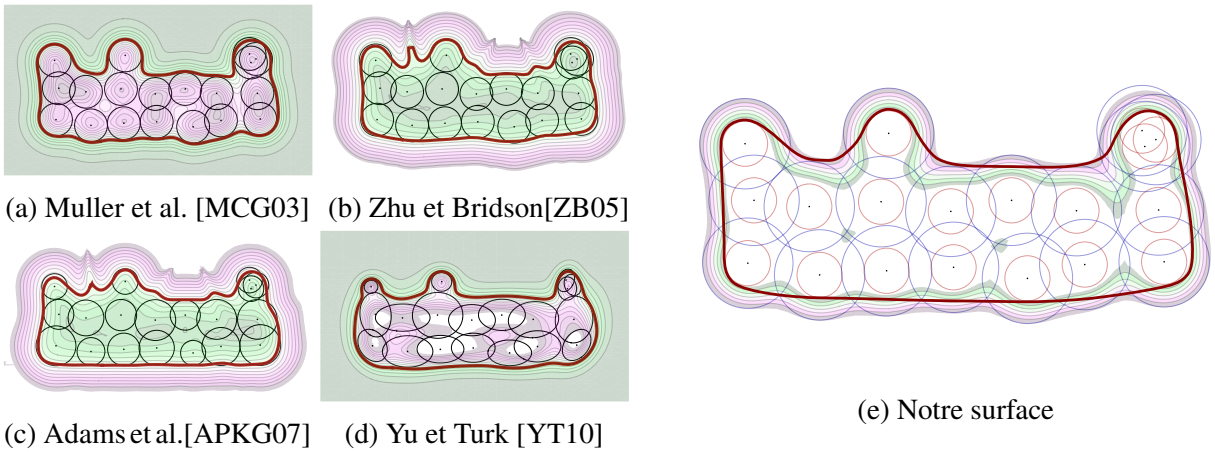
En conclusion, les surfaces implicites ont leurs avantages et leurs inconvénients, notamment :

+ Simplicité de la définition par somme de noyau.
+ Continuité de la représentation permettant de calculer plus aisément le gradient, la courbure, etc.
- Difficulté d'obtenir une surface plane.
- Besoin d'une structure supplémentaire pour stocker des valeurs de surface (couleur, température, dynamique de surface, etc.)

C'est pourquoi nous avons utilisé cette définition afin de construire des contraintes pour reconstruire itérativement notre surface par points (plus de détails à la section 3.2.2). Elle nous permettait non seulement d'avoir des contraintes lisses (améliorant nettement la convergence de notre surface par point), mais également d'approximer des valeurs des normales grâce au gradient. La figure 3.2 montre comment nous avons utilisé cette formulation implicite afin de définir des contraintes et comment cela permet d'obtenir des surfaces plus planes par rapport aux modèles utilisant uniquement une surface implicite.

3.2.2 Surface par points

Les surfaces par points sont des surfaces définies par un ensemble de points orientés (avec des normales) ou non. Ces points ne sont connectés d'aucune façon, par opposition aux maillages. Cela a l'avantage d'éliminer la nécessité de détecter les changements de topologie et de procéder à des remaillages, des opérations très fréquentes en simulation de fluide et particulièrement enclines aux erreurs dues à la difficulté de mettre au point des algorithmes rigoureux et robustes. Cependant, les maillages ont l'avantage qu'il est plus facile d'évaluer les propriétés de la surface telles que les normales et les courbures. Sans ces relations entre sommets définies explicitement par le maillage, on



(a) Muller et al. [MCG03] (b) Zhu et Bridson[ZB05]

(c) Adams et al.[APKG07] (d) Yu et Turk [YT10]

(e) Notre surface

Figure 3.2 – Schémas 2D des différentes méthodes de reconstruction de surface utilisant des modèles de surfaces implicites. Les surfaces implicites peinent à approximer les surfaces planes. Notre méthode utilise plutôt une représentation implicite afin de définir des contraintes, lui laissant plus de liberté afin de converger vers une surface plus plane.

doit donc utiliser d'autres méthodes afin d'évaluer les propriétés de la surface, que ce soit pour en faire le rendu ou évaluer les endroits de haute courbure pour y injecter de la turbulence.

Méthode des moindres carrés. Afin d'évaluer ces propriétés sur un ensemble de points, nous avons plutôt eu recours à la méthode des moindres carrés. Soit un système linéaire $M\chi = \beta$ où M est une matrice $n \times m$, $\chi \in \mathbb{R}^n$ et $\beta \in \mathbb{R}^m$ n'ayant pas de solution pour χ (i.e. β n'est pas dans l'espace des colonnes de M), la méthode des moindres carrés permet de trouver le vecteur χ tel que $M\chi$ est la meilleure approximation de β. Plus formellement, $\hat{\chi}$ constitue une solution au sens des moindres carrés de $M\chi = \beta$ si

$$||\beta - M\hat{\chi}|| \leq ||\chi - M\chi||$$

pour tout $\chi \in \mathbb{R}^n$. $M\hat{\chi}$ constitue donc la projection de β dans l'espace des colonnes de M et par conséquent $\beta - M\hat{\chi}$ est orthogonal à toutes les colonnes de M. Ainsi, on a $\mathbf{m}_i \cdot (\beta - M\hat{\chi}) = 0$ pour toute colonne $\mathbf{m}_i$ de M et $M^T(\beta - M\hat{\chi}) = \mathbf{0}$. On appelle "équations normales" de $M\chi = \beta$ les équations formées par le système $M^T M\chi = M^T\mathbf{b}$. Tout $\hat{\chi}$ satisfaisant $M^T M\chi = M^T\beta$ constitue une solution au sens des moindres carrés à $M\chi = \beta$. Si $M^T M$ est inversible, alors $\hat{\chi} = (M^T M)^{-1}M^T\mathbf{b}$. On peut également avoir recours à une factorisation QR ou une méthode itérative pour résoudre $\hat{\chi}$ afin d'éviter des erreurs numériques et les coûts de calcul élevés que peuvent entraîner l'inversion de matrices de

grande dimension.

En obtenant une formulation continue plutôt que discrète, il devient aisé d'obtenir des informations sur la surface obtenue, telles que la pente, la courbure, etc. Puisqu'on veut obtenir ces informations de manière localisée sur la surface, on utilise plutôt une version de la méthode des moindres carrée biaisée vers la région d'intérêt, soit la méthode des moindres carrés mobile (*Moving Least Squares*). Cette méthode est largement utilisée en informatique graphique et en vision tridimensionnelle afin de reconstruire une surface à partir d'un nuage de points. Dans cette formulation, on résout $\hat{\chi}$ dans un système de coordonnées local centré à $\mathbf{p}$ où $\mathbf{p}$ est la position du point de surface dont on veut approximer la fonction. Dans les moindres carrés, on minimise

$$||\chi - M\hat{\chi}|| = \sum_i (\beta_i - \mathbf{m}_i^T \chi)^2$$

. Les moindres carrés pondérés accordent plus d'importance à certaines données, c'est-à-dire

$$\sum_i (w_i(\beta_i - \mathbf{m}_i^T \hat{\chi}))^2 = ||W\beta - WM\hat{\chi}||$$

où W est une matrice diagonale avec $W_{ii} = w_i$.

Plus concrètement, soit un ensemble de points définissant une surface $S = \{\mathbf{p}_i | i \in [0,n]\}$ avec $\mathbf{p}_i \in \mathbb{R}^3$. Supposons que l'on veuille approximer la normale au point $\mathbf{p}_i$ par approximation d'un plan par les points dans un voisinage autour de $\mathbf{p}_i$. Il faut trouver les paramètres a, b, c définissant le plan $z = f(x,y) = ax + by + c$ qui minimise la somme pondérée des carrés des distances entre les points et le plan. Puisqu'on veut évaluer ce plan à $\mathbf{p_i}$, on utilise un système centré à $\mathbf{p}_i$ où la position de $\mathbf{p}_i$ dans ce système de coordonnées est donc $(x_i, y_i, z_i) = (0,0,0)$ et on définit l'axe des z par le gradient de la fonction de contrainte à $\mathbf{p}_i$ vue dans la section précédente. Le système correspondant est donc,

$$\begin{bmatrix} w_1 & 0 & ... & 0 \\ 0 & w_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & w_n \end{bmatrix} \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ & ... & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} w_1 & 0 & ... & 0 \\ 0 & w_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & w_n \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ ... \\ z_n \end{bmatrix}$$

Et on a donc

$$(WM)^T WM = \begin{bmatrix} \sum_j w_j^2 x_j^2 & \sum_j w_j^2 x_j y_j & \sum_j w_j^2 x_j \\ \sum_j w_j^2 x_j y_i & \sum_j w_j^2 y_j^2 & \sum_j w_j^2 y_j \\ \sum_j w_j^2 x_j & \sum_j w_j^2 y_j & \sum_j w_j^2 \end{bmatrix} \quad \text{et} \quad (WM)^T W\beta = \begin{bmatrix} \sum_j w_j^2 x_j z_j \\ \sum_j w_j^2 y_j z_j \\ \sum_j w_j^2 z_j \end{bmatrix}$$

Ce qui nous permet de calculer $(a \;\; b \;\; c)^T = ((WM)^T WM)^{-1}(WM)^T W\beta$ et d'obtenir la normale au point $\mathbf{p}_i$

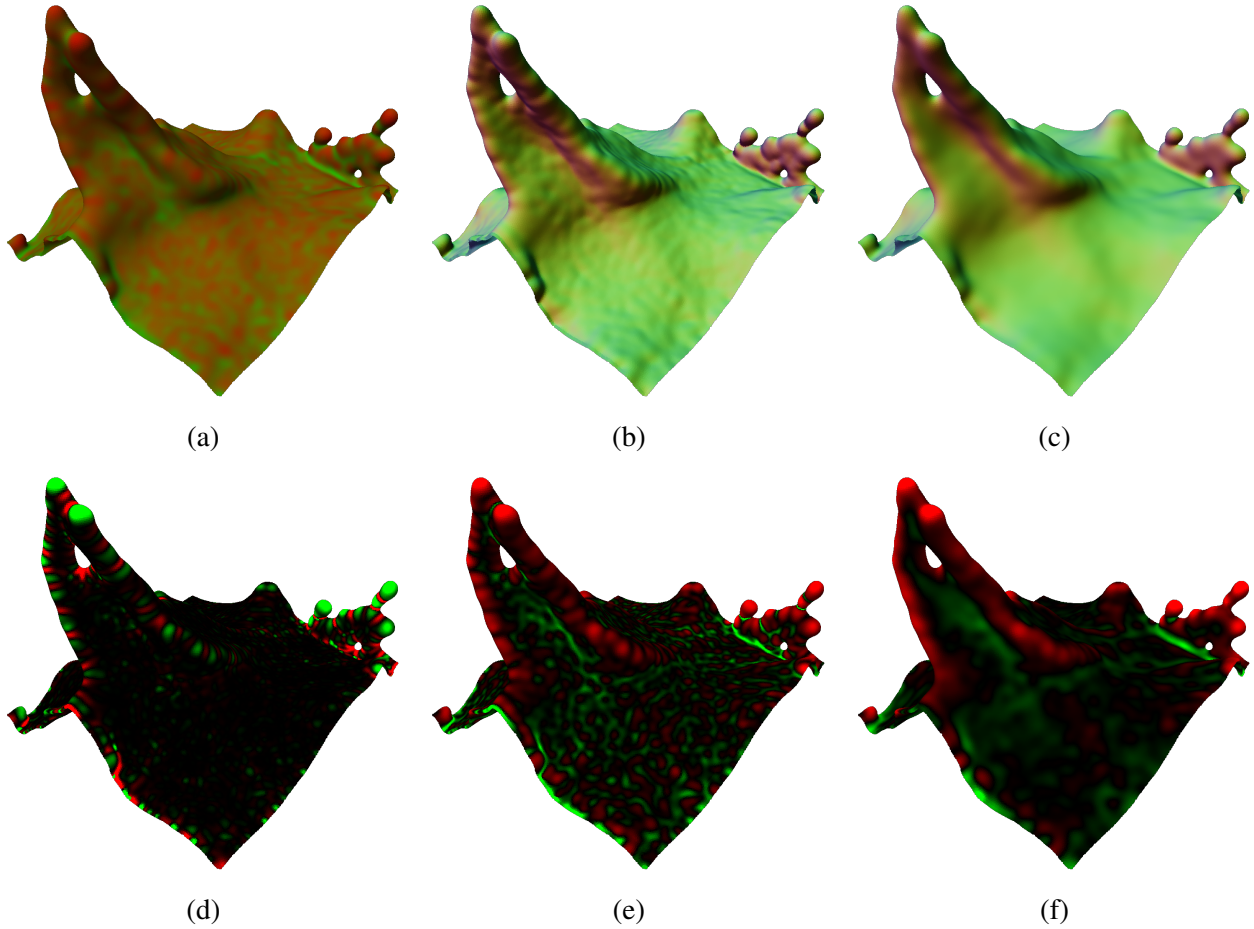
$$\mathbf{n}_i = \frac{\mathbf{v}_i}{|\mathbf{v}_i|} \quad \text{où} \quad \mathbf{v}_i = \begin{pmatrix} a \\ b \\ -1 \end{pmatrix}.$$

Bien sûr, on fait subir la transformation inverse à cette normale afin d'obtenir la normale en espace monde et non dans le système de coordonnées centré à $\mathbf{p}_i$.

On peut bien sûr utiliser la méthode des moindres carrés sous d'autres hypothèses que celle du plan, par exemple sous l'hypothèse d'une parabole. On obtiendrait alors les paramètres décrivant la parabole et l'on pourrait ainsi approximer non seulement la normale, mais également la courbure moyenne et gaussienne. Cependant, un problème que nous avons observé avec la méthode des moindres carrés est sa sensibilité à l'erreur. Un trop petit nombre d'échantillons ou de petites erreurs sur la position des points peut considérablement modifier les résultats obtenus par la méthode, en particulier lorsqu'on tente d'approximer une courbe polynomiale de degré n élevé. Ceci s'explique par le fait que réduire les échantillons ou augmenter le degré n du polynôme ajoute des degrés de liberté afin de réduire la somme des carrés des erreurs.

3.3 Turbulence de surface

Afin d'ajouter une dynamique à la surface par points reconstruite, nous avons utilisé une méthode d'intégration numérique afin de simuler l'équation d'onde. Nous verrons, dans cette section, un aperçu cette équation et de quelques méthodes d'intégration numérique, les détails d'implémentation étant abordés au chapitre 4.



(a) (b) (c)

(d) (e) (f)

Figure 3.3 – (a) Niveau ϕ dans la bande de contraintes (rouge = 0, vert = 1). (b) Gradients $\nabla\phi$. (c) Normales. (d) Courbure gaussienne évaluée avec les courbes de niveau. (e) Courbure moyenne évaluée avec les courbes de niveau. (f) Notre approximation de la courbure moyenne par différences de positions. Puisque les points de notre surface n'ont pas tous la même valeur de ϕ comme illustré dans (a), les résultats obtenus en (b), (d) et (e) contiennent du bruit basse fréquence et sont inconsistants avec la surface par points. La méthode des moindres carrés mobile (voir section 3.2.2) par rapport au plan nous permet de calculer fidèlement la normale (c) ainsi que la courbure (f) par différence de positions par rapport à la normale.

3.3.1 Équation d'onde

En physique des liquides, il y a deux types d'ondes surfaciques ; les ondes de gravité (influencées principalement par la gravité sur les ondes de basses fréquences) et les ondes supercifielles (influencées principalement par la tension superficielle sur les ondes de hautes fréquences). Or, l'un des éléments complètement négligés par l'équation de Navier-Stokes standard est la force occasionnée par la tension superficielle. La tension de surface d'un liquide est la force exercée par celle-ci afin de minimiser son contact avec l'air (i.e. de minimiser son aire). Au niveau moléculaire ceci s'explique par le fait que les molécules d'eau ont une plus grande attraction envers les autres molécules d'eau qu'envers les molécules d'air. C'est cette tension superficielle qui est responsable des effets d'onde qu'on peut apercevoir sur la surface de l'eau, par exemple lorsqu'une goutte tombe dans un bassin. La propagation de ces ondes est décrite par l'équation différentielle partielle de second degré suivante

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h \tag{3.19}$$

où t est le temps, h est la hauteur de l'onde, ∇^2 est l'opérateur laplacien et c est une constante. Dans notre article, chaque point de surface possède une valeur de hauteur h de vague et nous utilisons une méthode d'intégration numérique (plus spécifiquement la méthode symplectique d'Euler) afin de résoudre l'équation d'onde et ainsi propager les valeurs de hauteur h d'onde sur la surface.

3.3.2 Méthodes d'intégration numérique

En simulation numérique, on cherche généralement à résoudre des problèmes de Cauchy (ou problème de valeur initiale). Ce type de problème se définit par deux éléments : la valeur initiale décrivant un système évalué à t_0 et les équations différentielles ordinaires (avec une seule variable indépendante t) permettant de résoudre comment le système évolue par rapport à t. Autrement dit, soit

une fonction définie et dérivable $\mathbf{x}(\cdot)$ inconnue, on tente d'évaluer $\mathbf{x}(t)$ étant donné

$$\begin{cases} \frac{d\mathbf{x}}{dt} = f(t, \mathbf{x}(t)) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases}$$

Il est généralement impossible de trouver les solutions à ce type de problème analytiquement et on a donc plutôt recours à des méthodes numériques. Un grand nombre de ces méthodes reposent sur des approximations basées sur l'expansion de la série de Taylor. Cette série permet d'évaluer la valeur d'une fonction au voisinage de t par une série entière construite à partir d'une fonction f indéfiniment dérivable et de ses dérivées successives, et est définie par

$$f(t+\Delta t) = \sum_{n=0}^{\infty} \frac{\Delta t^n}{n!} f^{(n)}(t) = f(t) + \Delta t f'(t) + \frac{\Delta t^2}{2!} f^{(2)}(t) + \frac{\Delta t^3}{3!} f^{(3)}(t) + \frac{\Delta t^4}{4!} f^{(4)}(t) + \ldots \quad (3.20)$$

Nous verrons certaines de ces méthodes dans cette section.

Méthode explicite d'Euler. Si l'on tronque la série de Taylor à l'ordre 1, on obtient la formule explicite d'Euler

$$\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \Delta t f(t_0, \mathbf{x}(t_0)) \, . \quad (3.21)$$

On constate que cette approximation correspond aux termes de la série de Taylor tronquée à partir de la dérivée seconde et donc a une erreur d'ordre $O(\Delta t^2)$. L'erreur est donc nulle seulement si $\frac{d^2\mathbf{x}}{dt^2} = 0$ ou $\Delta t = 0$. Autrement, elle est linéairement proportionnelle au pas de temps Δt puisque

$$1 \text{ pas de } \Delta t \quad \Rightarrow \quad \text{erreur } \Delta t^2 \quad (3.22)$$

$$2 \text{ pas de } \tfrac{\Delta t}{2} \quad \Rightarrow \quad \text{erreur } 2\left(\tfrac{\Delta t}{2}\right)^2 = \left(\tfrac{\Delta t^2}{2}\right) \quad (3.23)$$

$$k \text{ pas de } \tfrac{\Delta t}{k} \quad \Rightarrow \quad \text{erreur } k\left(\tfrac{\Delta t}{k}\right)^2 = \left(\tfrac{\Delta t^2}{k}\right) \, . \quad (3.24)$$

En général, en simulation, on veut minimiser l'erreur tout en ayant un pas de temps suffisamment grand pour éviter les calculs inutiles. Mais encore plus important est la stabilité du système. Comme

nous le verrons au chapitre 4, notre méthode a comme objectif de réintroduire des détails de hautes fréquences dans les endroits où ceux-ci auraient été perdus par la simulation initiale. Puisque notre méthode est déjà une approximation afin de produire des résultats visuellement plausibles, nous nous soucions moins de l'erreur, mais davantage de la rapidité de la méthode d'intégration, de la stabilité du système et de la conservation d'énergie.

Méthode implicite d'Euler. Cette méthode a une erreur de même ordre que la précédente (premier ordre), mais utilise plutôt une formulation implicite qui tente d'évaluer l'état suivant étant donnée la dérivée de la fonction à l'état suivant. C'est à dire,

$$\mathbf{x}(t_0+\Delta t) = \mathbf{x}(t_0)+\Delta t f(t_0+\Delta t,\mathbf{x}(t_0+\Delta t)) \tag{3.25}$$

$$\mathbf{x}(t_0+\Delta t)+\Delta t f(t_0+\Delta t,\mathbf{x}(t_0+\Delta t)) = \mathbf{x}(t_0)\ . \tag{3.26}$$

Or, puisque l'état suivant est inconnu (c'est ce qu'on cherche à déterminer), on obtient un système linéaire à résoudre. Cette méthode est inconditionnellement stable, mais perd généralement beaucoup d'énergie en plus d'être coûteuse (due à la nécessité de résoudre un système linéaire).

Méthode symplectique d'Euler. La méthode symplectique (ou semi-implicite d'Euler) s'applique à des paires d'équations différentielles, où chacune est évaluée par la méthode d'Euler où l'une des évaluations utilise l'état courant et l'autre utilise l'état suivant sans nécessiter de résolution de systèmes linéaires. Par exemple, soient les équations de Cauchy permettant de décrire l'évolution des vélocités $\mathbf{v}$ par rapport au temps t et des positions $\mathbf{x}$ par rapport au temps t

$$\begin{cases} \frac{d\mathbf{v}}{dt} = f(t,\mathbf{v}(t)) \\ \mathbf{v}(t_0) = \mathbf{v}_0 \end{cases} \quad \text{et} \quad \begin{cases} \frac{d\mathbf{x}}{dt} = f(t,\mathbf{x}(t)) = \mathbf{v}(t) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases}$$

alors on résout les nouvelles vélocités et positions par

$$\mathbf{v}(t_0+\Delta t) = \mathbf{v}(t_0)+\Delta t f(t_0,\mathbf{v}(t_0))\ , \tag{3.27}$$

$$\mathbf{x}(t_0+\Delta t) = \mathbf{x}(t_0)+\Delta t f(t_0+\Delta t,\mathbf{x}(t_0+\Delta t)) = \mathbf{x}(t_0)+\mathbf{v}(t_0+\Delta t)\ . \tag{3.28}$$

Comme on peut le voir, cette formulation utilise à la fois le modèle explicite (équation 3.27) et

implicite (équation 3.28), mais il n'est pas nécessaire de résoudre un système puisque la dérivée de l'état suivant est directement donnée par la résolution de la première équation. Par conséquent, la méthode symplectique a une plus grande région de stabilité que la méthode explicite d'Euler et une meilleure conservation d'énergie que la méthode implicite d'Euler tout en étant très simple et rapide à évaluer, en faisant un bon choix d'intégrateur numérique afin de résoudre les équations d'onde dans notre méthode.

Chapitre 4

Surface Turbulence for Particle-Based Liquid Simulations

Ce chapitre expose notre article dans la langue originale de publication. Cet article a été un effort de collaboration pour l'entièreté de l'article, mes contributions étant particulièrement autour de la construction et de la maintenance de surface. Les auteurs sont Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim *et* Derek Nowrouzezahrai. *L'article a été publié dans ACM Transactions on Graphics 2015.*

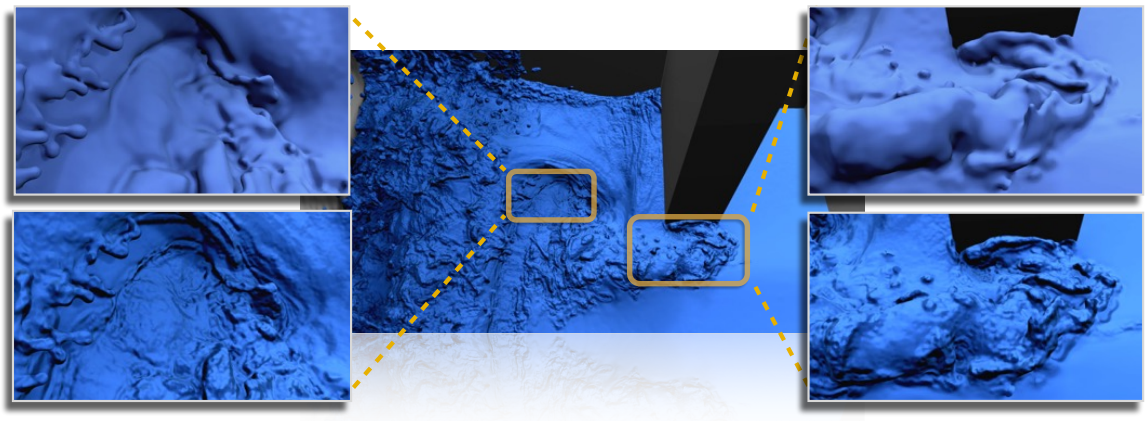


Figure 4.1 – We apply our turbulence model to a high-resolution FLIP simulation ($>12\times10^6$ particles). Zoom-ins compare the unmodified input surface (top) to our output (bottom). Even at high resolutions, the input simulation fails to resolve small scale details, which our method is capable of adding. In this extreme example, our entire post-process adds an overhead of roughly a third of the full simulation time.

Abstract

We present a method to increase the apparent resolution of particle-based liquid simulations. Our method first outputs a dense, temporally coherent, regularized point set from a coarse particle-based liquid simulation. We then apply a surface-only Lagrangian wave simulation to this high-resolution

point set. We develop novel methods for seeding and simulating waves over surface points, and use them to generate high-resolution details. We avoid error-prone surface mesh processing, and robustly propagate waves without the need for explicit connectivity information. Our seeding strategy combines a robust curvature evaluation with multiple bands of seeding oscillators, injects waves with arbitrarily fine-scale structures, and properly handles obstacle boundaries. We generate detailed fluid surfaces from coarse simulations as an independent post-process that can be applied to most particle-based fluid solvers.

4.1 Introduction

Simulating the behavior of fluids is a long standing problem that often requires visual details resolved to a very fine resolution. Simulating smoke and liquids are the two most common cases, and specialized approaches have been developed for each.

For *smoke* animation, Eulerian approaches are common. Here, performance scales with the underlying grid's resolution and resolving details of at fine scales quickly becomes prohibitive. *Fluid up-res* methods address this problem by applying fine-scale turbulence models atop coarser simulations, generating detailed results without explicitly simulating at a fine resolution [KTJG08, NSCL08, SB08].

These methods offer powerful means for art-direction, as users have the guarantee that the coarse behavior will not change when fine details are added.

In contrast, detailed *liquid* simulations are still performed at full resolution, regardless of whether a grid- or particle-based approach is used, with high-quality particle-based simulators, such as Fluid Implicit Particle (FLIP) [ZB05, Aut14] or Smoothed Particle Hydrodynamics (SPH) [MCG03, Nex14], having seen rapid, recent adoption.

We address the discrepancy with an *"up-res"* technique for particle-based liquid simulations. While a Eulerian *closest point turbulence* (CPT) method was recently developed for level set-based liquids [KTT13], it can only be applied to particle-based data by converting the data to an Eulerian grid, discarding many of the simulation's rich details. We instead add turbulent details *directly* to particles, solving the wave equation in a Lagrangian setting. We first convert a set of input particles from a

coarse liquid simulation into a high-quality, high-density surface point set. We then perform a wave simulation that adds high-frequency features to the liquid surface, in the form of bump or displacement maps. We use standard surfacing to obtain a detailed, high-resolution liquid surface.

To our knowledge, ours is the first comprehensive up-res technique for particle-based simulations, making the following contributions :

- robust, temporally coherent, meshless surface generation, that yields smooth, simulation-ready surfaces,
- smooth constraints to ensure that our surface remains spatially and temporally faithful to the underlying particle set,
- a robust, curvature-based method for initiating surface waves,
- a novel discrete Laplace operator that is provably well-suited for meshless point representations, in addition to
- an efficient simulation strategy that synthesizes details across scales onto our high-density surface.

Our method is agnostic to the source of particle data, and can be applied to FLIP, SPH, and position-based works [MMCK14].

4.2 Previous Work and Overview

Fluid simulation is a well-established area so, in addition to seminal works [FM96, Sta99, FF01, MCG03], we refer readers to Bridson's book [Bri08] and Ihmsen et al.'s STAR [IOS+14] for comprehensive surveys. Here, we focus primarily on areas most related to our work.

Fluid Up-res. Thuerey et al. [TKP13] surveys recent fluid up-res methods which efficiently increase the apparent spatial resolution of a coarse simulation without altering the low-frequency behavior. As noted in Section 4.1, these methods have been most effective in smoke simulations [KTJG08, NSCL08,

SB08, NCZ+09, HMK11]. Several works have also addressed the related problem of synthesizing frequency-controlled textures on moving surfaces [YNBH09].

Earlier attempts to apply up-res algorithms to liquids had limited success, both in Eulerian [NSCL08] and Lagrangian [YZC12, SZZW14] formulations, since they focus on increasing the resolution of the velocity field. As noted by Kim et al. [KTT13], the turbulence on a free surface is only loosely coupled to the fluid velocity field. Lab experiments show that surface waves tend to propagate much faster than the velocities of the underlying fluid would suggest. We thus choose to evolve a high-resolution simulation over the liquid surface to obtain more convincing results.

While CPT [KTT13] works well for Eulerian liquids, no comprehensive Lagrangian up-res method exists. This is unfortunate, because the ad-hoc methods developed in industry [BLMB13] show that there is substantial interest in such techniques.

Several works have explored how to guide liquids to meet artistic goals [SY05, PHT+13]. Nielsen and Bridson [NB11] use a low-frequency "guide shape", extracted from a coarse FLIP simulation, as the boundary condition for a thin, secondary, high-resolution simulation applied near the liquid boundary. Our work differs from this approach in two ways : first, we preserve the entire frequency content of the coarse simulation, including important quantities such as the silhouettes. Second, we add entirely new dynamics to the surface using a wave simulation. As such, our algorithm can complement such "guide shapes" approaches, especially since it is applied as a standalone post-process.

Surface Tracking. The importance of surface-only simulations has become increasingly clear in recent works on explicit surface tracking [TWGT10], and methods that use them [YWTY12]. Wojtan et al. [WMFB11] survey these recent developments. Instead of explicitly modeling the fluid surface and carefully incorporating it into the core simulation, we propose a post-process that can be applied directly to any coarse particle simulation, remaining *fully decoupled* from the simulator that generated the data. Moreover, maintaining a surface mesh throughout the simulation is cumbersome and often relies on external geometry processing tools. Our meshless approach is self-contained, simplifying implementation. While we create a mesh for rendering, meshing artifacts do not degrade the stability or robustness of the underlying simulation.

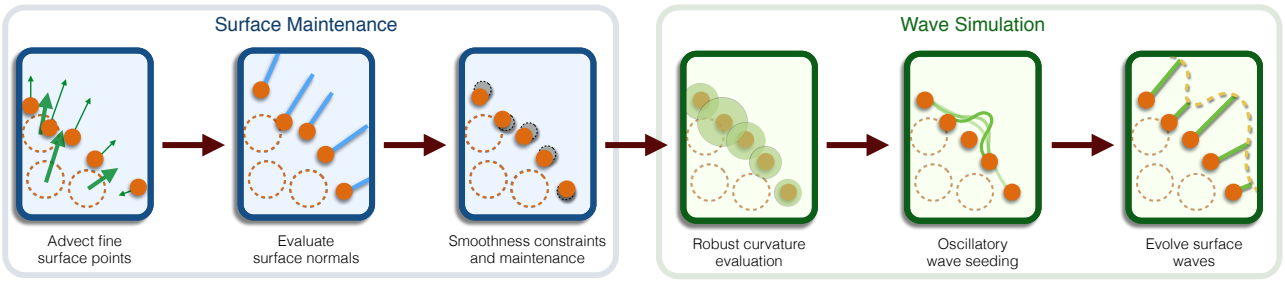


Figure 4.2 – An overview of the different steps of our algorithm, performed for each frame of coarse input data. The fine surface points (solid circles) are evolved on the surface of the input coarse particles (dashed circles). The overview in Section 4.2 details each stage of this diagram.

Inspired by optimization work for liquid surfacing [Wil08, BGB11], we constrain our final surface to lie in a band around the input surface. We extend ideas from Eulerian level sets and meshes to particle-based surfaces. While these works focus on generating geometry for rendering, we improve their smoothness and temporal regularity to make them suitable for simulation.

Wave Simulation. Many works consider wave simulations. From the linear wave equation [KM90], and related shallow-water models [WMT07], to bi-Laplacian variants [YWTY12] and the *iWave* [Tes08]. We use the linear wave equation and rely on dispersion from stretching induced by the underlying advection. As with all these previous works, we manually set an average propagation speed for our surface waves.

On the other hand, *wave particles* represent surface waves [YHK07, Cor08] with a dense set of advected points. These methods have difficulty with the complex, topologically-varying surfaces that we treat, requiring many more wave particles to represent the details we achieve with our approach.

Point-based Simulation. Our work is related to point-based techniques, but unlike previous works that deal with static point sets [ZPKG02, ABC$^+$03] or dynamic surfaces for rendering [GGG08], our input particles represent a volume where every particle corresponds to a quantity of liquid. Point-based rendering treats each sample as a (possibly noisy) surface point, and previous point-based simulations [MMR13, JK13] operate on static point sets, and are thus not appropriate for dynamic particle sets from liquid simulations.

Overview. Figure 4.2 is a visual overview of our method. Given an input sequence of particles representing a liquid volume (*coarse particles*, dashed circles), we first construct a dense point set along the liquid interface (*fine surface points*, solid circles ; Sections 4.3.2 & 4.3.3). We smooth and regularize these points to support point-based simulation (Section 4.3.4). Using per-point normals and displacement values we call *wave values* (green lines ; Section 4.4.2), we perform a high-resolution wave simulation (green curve ; Section 4.4.3) over the surface. We output the final detailed surface as a bump or height map over the high-density point set, or as a displaced point set. These points can be splatted directly or tessellated for rendering.

4.3 Surface Construction and Maintenance

We construct a dense point set that represents the liquid's surface, and maintain a level of smoothness and regularity necessary for point-based wave simulation. We describe our novel smooth band constraint, which controls the surface's behavior, and ensures coherence between the surface points and underlying simulation. Unlike level set or mesh-based surface tracking [OF03, WMFB11], our fluid surface is represented exclusively by oriented points i, each with a position $\mathbf{x}_i$ and normal $\mathbf{n}_i$.

4.3.1 Neighborhood Relationships

To ensure that our surface points behave as a unified manifold, we draw upon work in meshless simulation (e.g., [IOS$^+$14]), taking advantage of the neighborhood relationships between point pairs. Our high-resolution surface points $\mathbf{x}_i$, and coarse input particles $\mathbf{X}_i$, will affect each other across spatial scales. Specifically :

- a coarse-scale length λ_c, obtained from the coarse particle simulator (e.g., the grid cell size in a FLIP solver), and
- a fine-scale length λ_f, a user parameter controlling the separation between points of the detail-enhanced surface.

We use λ_c for operations related to the underlying fluid, such as surface advection (Section 4.3.2) and curvature evaluation (Section 4.4.1), and λ_f for intrinsic surface operations, including point distribution

regularization (Section 4.3.4) and wave evolution (Section 4.4.3). We use isotropic kernels to weight the effect of particles on their neighbors. Unless specified otherwise, we use a simple triangular kernel :

$$K_i^\delta(\mathbf{x}_j) = 1 - ||\mathbf{x}_i - \mathbf{x}_j||/\delta, \text{ if } ||\mathbf{x}_i - \mathbf{x}_j|| < \delta \,;\, 0, \text{ otherwise}\,, \tag{4.1}$$

where δ is the local neighborhood radius. We normalize the weighting kernel according to the local density around a point j,

$$\rho_j^\delta = \sum_{k \in \mathscr{F}} K_j^\delta(\mathbf{x}_k)\,, \tag{4.2}$$

where $\mathscr{F}$ denotes the set of all surface points. This local normalization eliminates any bias introduced by potential non-uniformities across local point densities, and so the local weight is $w_i^\delta(\mathbf{x}_j) = K_i^\delta(\mathbf{x}_j)/\rho_j^\delta$. This density normalization is especially important at the finest scale, where point distribution variance is highest.

We also normalize the weighting so the contributions sum to unity, so our final weight of point j for point i is

$$W_i^\delta(\mathbf{x}_j) = w_i^\delta(\mathbf{x}_j) \Big/ \sum_{k \in \mathscr{F}} w_i^\delta(\mathbf{x}_k)\,. \tag{4.3}$$

In practice, we only sum over particles in finite neighborhoods due to the local support of K. The weights also apply naturally to coarse particles by exchanging $\mathscr{F}$ for $\mathscr{C}$, the set of all coarse particles.

4.3.2 Surface Initialization and Advection

We create the initial set of fine-scale surface points in two steps. We first create an initial point set by centering spheres of radius λ_c around each coarse particle, sampling points uniformly on these spheres, and only retaining points that do not fall inside any other sphere's volume. This process results in a very rough, non-smooth, fine-scale surface point distribution that we regularize in a second step (see Section 4.3.4). For any subsequent frame n, fine surface points $\mathbf{x}_i^n$ are obtained by simply advecting points $\mathbf{x}_i^{n-1}$ from the previous frame. The updated surface point position is a weighted sum

of the displacements of neighboring coarse particles,

$$\mathbf{x}_i^n \leftarrow \mathbf{x}_i^{n-1} + \sum_{k \in \mathscr{C}} W_i^{2\lambda_c}(\mathbf{X}_k)\,(\mathbf{X}_k^n - \mathbf{X}_k^{n-1}). \tag{4.4}$$

We can modify the neighborhood size used for advection according to how closely we want fine-scale surface points to track coarse particles ; in practice, a neighborhood radius of $2\lambda_c$ yields smoothly advected fine-scale surfaces. We use this value for all of our results.

After advection, fine-scale points may no longer be located in the vicinity of the coarse particles, so the fine-scale surface behavior may deviate from the dynamics of the underlying coarse simulation. Maintaining a correspondence between the input (coarse) and output (fine) fluid behavior is essential for predictable artistic control, so we next devise a surface constraint that imposes this guarantee.

4.3.3 Surface Constraints

A fundamental component of our approach is the generation of a smooth, temporally coherent, high-resolution output surface that does not need to explicitly track manifold connectivity. To accomplish this, we devise an implicit representation that constrains the position of the fine-scale surface points.

Our method is motivated by Williams [Wil08] : First, two concentric spheres of radius r and R are centered at each coarse particle, where R is the larger of the two. During surface evolution, the fine-scale surface points are constrained to remain in the volume between the union of all outer spheres and the union of all inner spheres. As depicted in Figure 4.3, this volume region forms "bands" around the coarse particles, delimiting the region where the advected fine scale surface is allowed to exist. This constrains surface points to regions not too far from, nor too close to, the existing coarse particles.

The r and R parameters affect the final output appearance. Small values create surfaces that more closely resemble the coarse simulation, but increase bumpiness. Large values can create surfaces that deviate significantly from the coarse simulation and exhibit over-smoothing. While they can be manually adjusted, we found that $R = \lambda_c$ results in a reasonably smooth surface relative to the underlying simulation and $r = R/2$ leaves sufficient space for fine particles to evolve without closely approaching the coarse particles.

Williams uses the spheres as constraints in a thin-plate energy optimization to represent the surface, but it is unclear how to adapt this to a meshless setting without performing a costly intermediate meshing. Moreover, a projection onto the surface formed by the union of spheres is difficult due to discontinuities in the first derivatives, and an orthogonal projection would lead to discontinuities that make the surface unsuitable for wave simulation (Section 4.4.3).

To solve this problem, we instead project onto an implicit metaball-like formulation [Bli82] that leads to smooth projection constraints better suited to the volumetric region between the inner- and outer-sphere unions. The efficacy of this strategy is shown in Figure 4.3. This novel *smooth band constraint* is constructed from an implicit function $\phi(\mathbf{y}) = g(f(\mathbf{y}))$ for an arbitrary point $\mathbf{y}$, where f is a standard metaball function and g is a rescaling function. There are many possible choices for f, but we obtained good results with

$$f(\mathbf{y}) = \sum_{i \in \mathscr{C}} f_i(\mathbf{y}) / \psi_i \quad \text{with} \quad f_i(\mathbf{y}) = \exp\left(-a|\mathbf{y} - \mathbf{X}_i|^2\right), \tag{4.5}$$

where ψ_i is the metaball density of the i-th coarse particle and a is a falloff parameter discussed below. The metaball density ψ_i differs from the local density ρ_i^δ in Equation 4.2, and is evaluated as

$$\psi_i = \sum_{j \in \mathscr{C}} D(||\mathbf{x}_i - \mathbf{x}_j||) \quad \text{with} \quad D(z) = \exp\left(-2\,(z/\lambda_c)^2\right). \tag{4.6}$$

Note that using different kernels for $f_i(\mathbf{y})$ and $D(z)$ provides the flexibility necessary to design sufficient constraints. Similar to Eqns. 4.1 to 4.3, we only consider local neighborhoods of surface points in the sums in Eqns. 4.5 and 4.6 due to the kernel's exponential fall-off. Unlike before, truncating the summation *does* introduce error, but we found that a neighborhood radius of $2R$ renders it negligible.

Finally, we impose an almost linear variation in ϕ, from 0 at the inner sphere to 1 at the outer sphere, as illustrated in Figure 4.3, using the following scaling function :

$$g(z) = \left(\sqrt{-\ln(z)/a} - r\right) \Big/ (R - r). \tag{4.7}$$

We solve for a value of a that, given two coarse particle centers less than $\mu = 3/2 \times R$ units apart, will

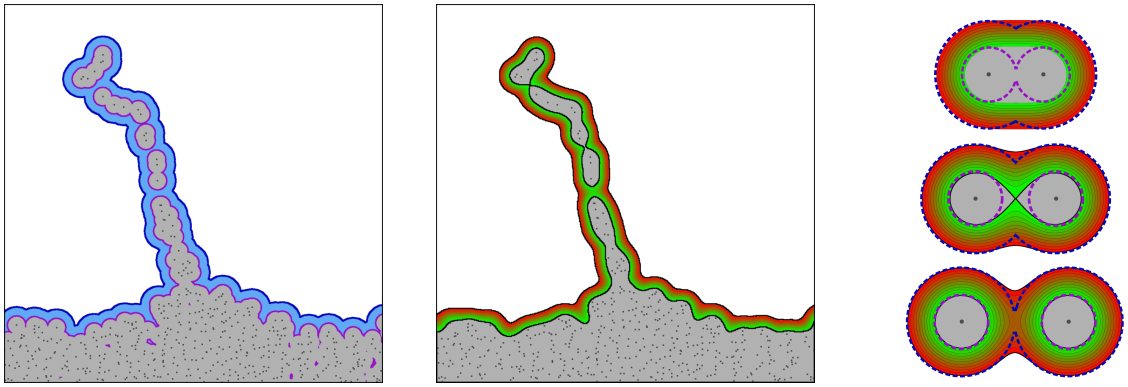


Figure 4.3 – Williams' constraint circles (left, right dashed circles) and our smooth band constraint (center, right color gradient). Our constaints are smoother and define values in the interior that vary linearly in space, permitting simpler projections.

unify the inner sphere union constraint of the two particles as if their contribution resulted from the same component of the fluid surface :

$$a = \ln\left(2/[1+D(\mu)]\right) \Big/ \left((\mu/2)^2 - r^2\right). \tag{4.8}$$

Figure 4.3 (bottom middle) is the case with particle centers exactly μ units apart. All our results use this value of a, and its associated μ.

Since our band constraint function ϕ is smooth, we can use its normalized gradient $\mathbf{g}$ to determine a reasonable projection direction when a particle that exits the constraint region. Additionally, as the function is approximately linear with respect to the distance from the inner constraint, we can easily compute this projection as

$$\mathbf{x}_i \leftarrow \begin{cases} \mathbf{x}_i - (R-r)\cdot\phi(\mathbf{x}_i)\cdot\mathbf{g}_i & \text{if } \phi(\mathbf{x}_i) < 0 \\ \mathbf{x}_i - (R-r)\cdot(\phi(\mathbf{x}_i)-1)\cdot\mathbf{g}_i & \text{if } \phi(\mathbf{x}_i) > 1 \\ \mathbf{x}_i & \text{otherwise .} \end{cases} \tag{4.9}$$

This introduces some approximation error, so a surface point may still lie outside the constraint region after projection. However, at this stage, it keeps points sufficiently close to the constraint region.

4.3.4 Surface Smoothing and Regularization

We now have a high-density set of surface points, but their distribution must be improved (i.e. regularized) before they are suitable for surface wave simulation. This regularization proceeds in four stages : normal computation, normal regularization, tangent regularization, and point insertion and deletion.

Normal Computation : A smooth, artifact-free normal field is essential for the maintenance of our surface structures. We compute the normal at each fine-scale surface point using an averaged least-squares planar fit to the local gradient of ϕ (see Algorithm 1).

Regularization Along the Normal : To improve the smoothness of our surface, we displace each surface point along its newly computed normal direction. Given a point i and one of its neighbors j, we consider the plane $\Pi_{i,j}$ spanned by vectors $\mathbf{n}_i$ and $(\mathbf{x}_j - \mathbf{x}_i)$. We find the[1] circle in this plane that is equidistant to each point and orthogonal to both points' normals (see Figure 4.4). The projection of point i onto this circle is averaged over all its neighbors, and the point is then displaced along its averaged projected position.

This averaging is done in a neighborhood of radius λ_c, which pushes the points towards a surface that is consistent with the computed normal field. Since the normals vary smoothly, the resulting surface is also smooth. Denoting $\mathbf{n}_j^\star$ the normalized projection of $\mathbf{n}_j$ onto $\Pi_{i,j}$, the displacement of point i onto the circle is given by

$$\text{proj}_{i,j} = \mathbf{n}_i \frac{(\mathbf{n}_i + \mathbf{n}_j^\star) \cdot (\mathbf{x}_i - \mathbf{x}_j)}{2\,\mathbf{n}_i \cdot (\mathbf{n}_i + \mathbf{n}_j^\star)}, \tag{4.10}$$

so this regularization step is computed as

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j \in \mathscr{F}} W_i^{\lambda_c}(\mathbf{x}_j)\,\text{proj}_{i,j}\,. \tag{4.11}$$

[1] There is a unique solution (i.e. circle) that satisfies these constraints.

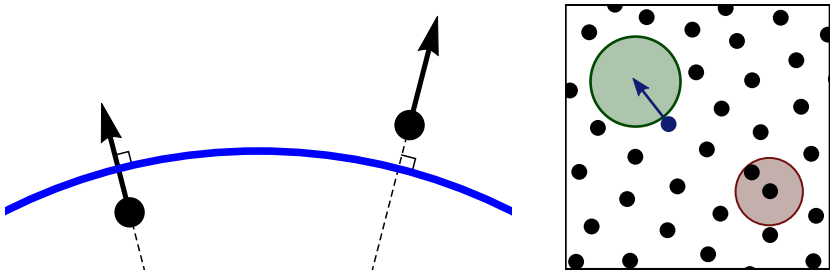


Figure 4.4 – Surface regularization shifts points along their normal towards circles consistent with the points' positions and normals, smoothing the surface. Right : A surface point is added to a low density region (green) and deleted from a high density region (red).

forall the *surface points i* **do**
  $\mathbf{n}_i \leftarrow \mathbf{g}_i$;
  compute tangent $\mathbf{t}_i^1$ and bi-tangent $\mathbf{t}_i^2$, orthonormal to $\mathbf{n}_i$;
  least square plane fitting in λ_c to the frame $[\mathbf{t}_i^1, \mathbf{t}_i^2, \mathbf{n}_i]$;
  $\mathbf{n}_i \leftarrow$ normal of plane;
forall the *surface points i* **do**
  $\mathbf{n}_i \leftarrow$ averaged normal in neighborhood λ_c;
  $\mathbf{n}_i \leftarrow$ normalize $(\mathbf{n}_i)$;

Algorithm 1: Steps for computing the normals.

Regularization Along the Tangents : Similar to previous works, we insert repulsion forces [Tur91] to improve the distribution of the surface points by driving them along their tangent directions. At each surface point, we compute the weighted direction *away* from its neighbors, and displace it in this direction. This averaging is performed in a local neighborhood λ_f around the point. Explicitly, this regularization step is computed at each surface point as

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + 0.5\,\lambda_f \sum_{j \in \mathscr{F}} W_i^{\lambda_f}(\mathbf{x}_j)\,\text{normalize}(\mathbf{x}_j - \mathbf{x}_i)\,, \tag{4.12}$$

where the 0.5 factor avoids moving two points to the same location.

Insertion and Deletion : The last regularization step adds and deletes surface points according to changes in the underlying simulation. Points are deleted when the local point density is too high. We detect this by looking for pairs of points that are closer than $3/4\,\lambda_f$, in which case the most recently

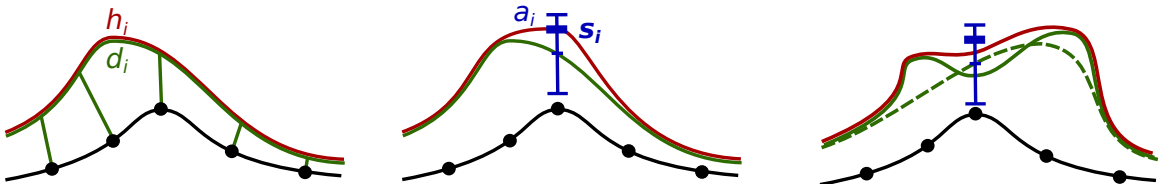


Figure 4.5 – Our wave seeding strategy. A wave moving from the left side of the simulation reaches the highlight region of the surface (left) ; seeding has yet to occur here, so the displayed (d_i, green) and internal (h_i, red) waves match. The underlying surface has high curvature at the center surface point (middle). We increase the oscillator amplitude (a_i) here from 0 to Δa, and compute a wave seeding value (s_i) from a cosine oscillator with amplitude a_i. We evolve the wave simulation and subtract the seeding values from the computed wave to obtain the new displaced wave value (right). The dashed green line shows what the displayed wave would have been if no wave seeding had occurred.

created point is deleted. Similarly, surface points are added when the local point density is too low, by Poisson disk sampling [Coo86]. For each point, we recompute the tangential direction as it was computed in the previous tangent regularization step. This generally points in the direction of lowest density, so we place a sphere of radius λ_f at a distance λ_f in this direction and search for neighboring points that fall inside the sphere. If none are found, the point density is too low and we create a point at the displaced sphere's center (Figure 4.4).

The three regularization steps are global operations that influence the *entire* surface. However, as they each consist of spatially local computations, we apply them several times to each animation frame until convergence. We used 30 iterations for the first frame and between 3 and 10 iterations per subsequent frame in all of our simulations. The number of iterations used for each scene is shown in Table 4.I. The accompanying video shows the uniform point density maintained by our surface operations on a deforming fluid surface.

4.3.5 Interactions with Obstacles

We have so far focused on the management of surface points in unobstructed flow, but special care must be taken when surface points approach obstacles. The regularization in Section 4.3.4 work best if the point distribution around each surface point is approximately uniform, which is not the case near obstacle boundaries.

Motivated by ghost and boundary [SB12, AIA+12] particles in SPH, we add ghost points for each

surface point located close to an obstacle by reflecting the neighboring points of the current point w.r.t. the obstacle. We then simply apply the regularization steps to both "real" and ghost point sets.

4.4 Turbulence Creation and Evolution

Section 4.3 detailed the creation of high-density point sets that represent the underlying coarse fluid simulation surface. We can now add turbulent details atop this surface, simulating waves on its points.

4.4.1 Curvature Evaluation

Detailed surface waves should appear in areas of high activity, e.g., regions that are merging or separating. Mean surface curvature is a good indicator for these regions [KTT13], as high curvature typically denotes areas that were under-resolved in the original simulation. We use waves to re-introduce these lost details. A common approach to compute curvature on a point surface is to use a second-order neighborhood fitting. This works well for smooth surfaces [WYLC13], but we are dealing with a turbulent surface. In this case, quadratic fittings can generate extreme and noisy curvatures that lead to instabilities over time.

We instead propose a new, robust alternative for evaluating curvature on point clouds. The curvature c_i at point i is defined as the signed distance of its neighbors to its tangent plane, easily obtained from the normal computed at each point (Section 4.3.4), which gives :

$$c_i = \sum_{j \in \mathscr{F}} W_i^{\lambda_c}(\mathbf{x}_j)\,(\mathbf{n}_i \cdot (\mathbf{x}_i - \mathbf{x}_j))\,. \tag{4.13}$$

This value is not identical to mean curvature, but we found it to be a very reliable criterion for wave generation. It is especially important for any such criterion to be thresholded in a meaningful way across discretizations. We derive practical thresholds $\{c_{\min}, c_{\max}\}$ by evaluating our curvature measure for two extremal fluid scenarios : a single drop and a thin sheet. For a surface of infinite point density, at distance λ_c from the coarse particles, our curvature measure in these two scenarios evaluates to $c_{\max} = 0.15\,\lambda_c$ for a single drop and $c_{\min} \approx 0.0771413\,\lambda_c$ for a thin sheet. These thresholds are very

useful for parametrizing simulations, as will be outlined in the following section. See Appendix A for threshold derivations.

4.4.2 Turbulence Creation

Data:

c_i	: surface curvature	c	: wave speed
$c_{\text{min,max}}$	: curvature thresholds	f_b	: base seeding frequency
a_i	: oscillator amplitude	f_o	: # of frequency octaves
Δa	: oscillator amplitude step size	W	: max. wave amplitude
A	: max. oscillator amplitude	F	: max. wave frequency
h_i	: internal wave height	t	: simulation time
v_i	: internal wave velocity	Δh_i	: laplacian
d_i	: displayed wave height	s_i	: seed value

forall the *surface points i* **do**
  $a_{\text{tmp}} \leftarrow 2\ \text{smoothstep}(|c_i|, c_{\min}, c_{\max}) - 1;$
  $\text{a}_i \leftarrow \text{clamp}(\text{a}_i + a_{\text{tmp}}\ \Delta a, 0, A);$

  $s_i \leftarrow 0\,;\ f \leftarrow f_b\,;\ a \leftarrow a_i;$
  for $\lambda = 1\,..\,f_o$ **do**
    $s_i \leftarrow s_i + a_i\ \cos(t\ c\ f);$
    $f \leftarrow 2\ f\,;\ a \leftarrow a/2;$
  $h_i \leftarrow d_i + s_i;$
forall the *surface points i* **do**
  $v_i \leftarrow v_i + c^2\,\Delta t\,\Delta h_i;$
  $h_i \leftarrow h_i + \Delta t\ v_i;$
  $d_i \leftarrow h_i - s_i;$
  $d_i \leftarrow \text{clamp}(d_i, -W, W);$
  $v_i \leftarrow \text{clamp}(v_i, -W\ F, W\ F);$

Algorithm 2: Pseudocode for wave evolution and seeding.

The simplest way to add turbulence to a surface is to add it directly to the wave heights. This can work well for grid-based methods [KTT13] where surface curvature varies smoothly with respect to grid size, but particle-based methods often contain large, abrupt curvature variations. For instance, when a single particle falls onto a flat water surface, a discontinuous wave seeding causes an abrupt visual change in height. Moreover, curvature is always high on isolated spherical droplets, causing

waves to be seeded uniformly over the entire droplet and creating non-realistic pulsing behaviors that do not conserve mass (see supplemental video).

A simple solution used in mesh-based simulations [YWTY12] is to turn off the wave seeding in regions of high curvature. This reduces artifacts from curvature discrepancies, but still produces abrupt changes when the seeding is toggled on and off in regions of near-maximum curvature. Moreover, it removes some of the waves caused by fine splashes characteristic of particle-based fluids, and limits the amount of new detail that is added to the surface.

We propose a different approach, outlined in Algorithm 2 and illustrated in Figure 4.5. High curvature regions seed new waves based on our curvature thresholds. In lines 2 and 3, we throttle seeding based on the curvature thresholds in Section 4.4.1. We seed with time-varying cosine oscillators (line 6), but these oscillations are never directly visualized. Instead, we apply the wave equation to them and only new waves that have *propagated out* from the cosine oscillations are visualized. This avoids double accumulation of wave values in regions of high curvature : once from their own oscillator and once from the waves generated by neighboring oscillators. This also causes waves to appear at the boundary of the seeding regions : for an isolated droplet, the seeding region has no boundary and, as no wave is seeded, the pulsing artifact is removed.

Our seeding is easy to implement : in addition to the wave value h_i used to solve the wave equation, we store a *displayed wave value* d_i. We store cosine oscillators separately in *seed values* s_i. At each step, we first add the seed values to the displayed wave values to obtain the wave values (line 8). We perform the wave simulation on h_i (lines 10, 11), and we subtract the seed values from h_i to recover d_i (line 12). The d_i values thus contain only the new features that propagated out from the seed values, in addition to features that have evolved from previous timesteps. The h_i are never visualized.

Finally, inspired by smoke up-res methods [KTJG08], our approach seeds features across multiple frequency octaves. To enable further control, the base seeding frequency f_b of the cosine oscillators, and the number of additional octaves f_o, are exposed as user parameters (lines 4 and 5). The accompanying video shows a comparison of our seeding method versus more direct approaches.

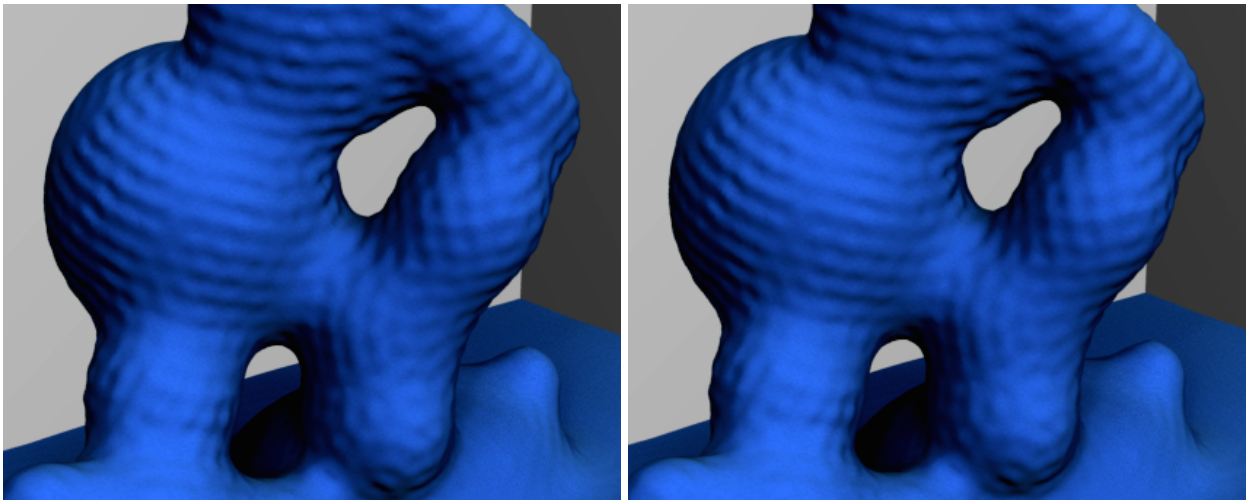


Figure 4.6 – Comparing wave propagation using the Laplace-Beltrami operator (left) and our simpler flat Laplace operator (right). The approximation error is negligible : even after 1000 simulation steps, there are no observable differences.

4.4.3 Turbulence Evolution

We evolve waves on the surface using the standard wave equation,

$$\partial_t^2 h = c\,\Delta h\,, \tag{4.14}$$

where c is the wave speed and Δ is the Laplace-Beltrami operator on the surface. We solve this equation explicitly with a symplectic Euler scheme that also requires the wave velocity v_i to be stored at each surface point. The linear wave equation, due to its dispersive nature, only approximates capillary surface wave behavior. Despite this, the linear wave model is widely used [TWGT10, CM10] and yields convincing results. Similar to these works, we add a small amount of diffusion to approximate wave dissipation, but this is not required to maintain stability.

Some previous works solve differential equations involving the Laplace-Beltrami operator on point surfaces [LLWZ12, LZ13], requiring the evaluation of surface curvature as well as functions of curvature. In our case, the curvature of the surface is small compared to the length λ_f where the wave equation operates, so we instead approximate the Laplace-Beltrami operator in Equation 4.14 with a simpler, flat Laplace operator. We avoid the metric tensor computations of Laplace-Beltrami, which are

both costly and difficult to robustly evaluate on point surfaces. Figure 4.6 compares our flat Laplace operator to Laplace-Beltrami, showing results that are indistinguishable even after 1000 simulation steps. Here, the approximation was 6 times faster to compute and gave a maximum height difference of only 1.4%. Appendix B further validates our approximation.

A common method [LZ13] for computing the flat Laplace operator on a point surface is to use a local quadratic least squares approximation of the function, and then use the derivatives of this approximation. Although this works with densely sampled surfaces, we generate waves at scales that can be of the same order as the distance between points. Only a few points can then be used for the quadratic approximation, which is imprecise and leads to instabilities over time. Our supplemental video and figure 4.7 illustrate such instabilities.

We instead compute the Laplace operator by using the tangent plane, which was previously obtained during surface normal computation (section 4.3.4). By projecting nearby points onto the tangent plane, the displacement defined by the wave values h_i becomes a function defined on the tangent plane. In this coordinate system, an affine approximation P of the function is first computed using standard least-square minimization. Subtracting the values of P on each neighboring point eliminates the zeroth- and first-order derivatives of the function, so the Laplacian can be evaluated directly as a weighted sum of discrete directional second-order derivatives :

$$\Delta h_i = \sum_{j \in \mathscr{F}} W_i^{2\lambda_f}(\mathbf{x}_j) \frac{4\left((h_j - P(\mathbf{x}_j)) - (h_i - P(\mathbf{x}_i))\right)}{||\mathbf{x}_i - \mathbf{x}_j||^2}. \tag{4.15}$$

We have found that a neighborhood radius size of $2\lambda_f$ works well. Appendix B shows how our operator approximates the Laplacian. To our knowledge, we are the first to introduce this discrete operator for computing the Laplacian on a meshless set of points. Our supplemental video and figure 4.7 show that it matches computations using the usual quadratic least squares approximation. Furthermore, since it uses an affine least squares fit instead of a quadratic fit, our operator remains stable even for waves at the highest frequency representable by the surface points. As such, it is particularly well suited to our meshless point representations, and we observed it was 2 times faster to evaluate than the traditional quadratic least squares Laplacian in our tests.

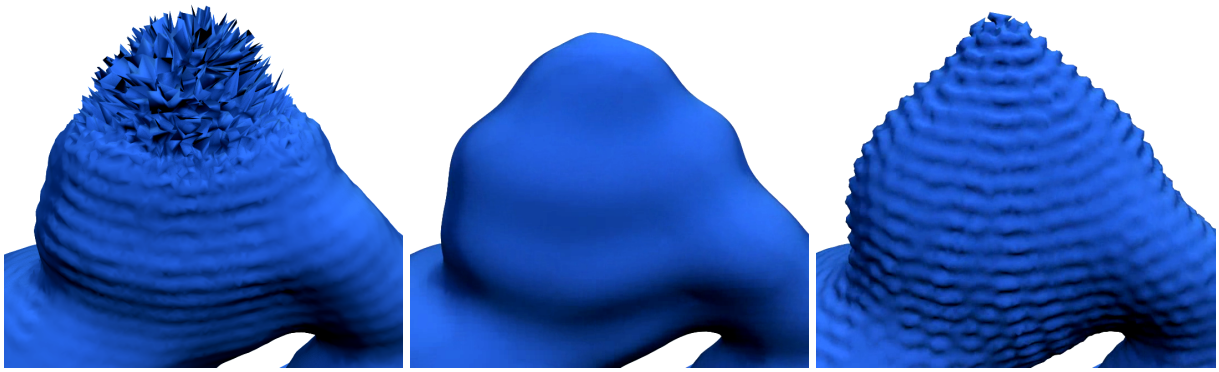


Figure 4.7 – Comparing the Laplacian computed with a least squares fit to our new discrete operator. When generating waves at a scale close to the limit of the point density, the least squares fit fails to isolate the desired wavelength and becomes unstable (note the undesirable small scale waves, left). For the same wave frequency, our discrete Laplace operator has no problem correctly treating the wave (middle). Our discrete operatore remains stable even when pushed to the Nyquist frequency limit of the discretization (right).

Scene	# particles	# surface points	Total (sec/frame)	Advection	Regularization	Curvature	Laplacian	Wave	Disk I/O
Dam Break	12500k	500k	85.4	8.6%	53.1% (10)	5.4%	2.9%	0.3%	2.0%
River	400k	280k	42.2	14.7%	48.7% (3)	9.9%	10.8%	0.8%	15.1%
Double Drop	1400k	350k	25.9	8.1%	57.0% (5)	5.4%	7.9%	1.0%	4.9%
Stir	390k	17k	1.5	15.0%	49.9% (5)	6.7%	4.1%	1.1%	4.6%
	390k	66k	5.4	10.8%	58.1% (5)	7.5%	6.0%	0.8%	4.8%
	390k	145k	15.2	8.7%	61.0% (5)	7.0%	9.8%	0.6%	3.9%

Tableau 4.I – Timings for the various steps of our algorithm. All performance statistics were computed on an Intel i7 quad core running at 3.4 GHz with 32GB of RAM. In the regularization column, the parenthesis indicate the number of regularization steps used per frame.

As explained in Section 4.3.5, ghost points are injected when surface points approach obstacles. These ghost points are also used during the wave computations, where the wave value on a ghost point is simply copied from its original "real" surface point. This results in a Neumann boundary condition on the obstacles that yields the expected wave reflections.

4.5 Results and Discussion

Up-resed Simulations. We apply our method to three coarse simulations, all of which generated using Houdini 13's FLIP solver. Total timings include disk I/O.

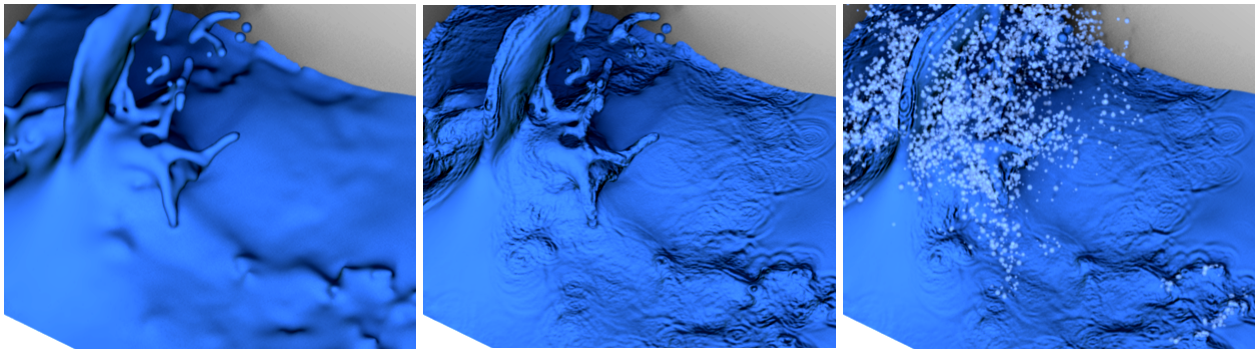


Figure 4.8 – We upres an input simulation (left, 1.4 million particles) with 400K surface points (middle), augmenting details over the bulk of the surface. We can also combine our results with other methods (i.e., [IAAT12]) to further increase the visual fidelity (right).

In Figure 4.1 and 4.12 we apply our method to a large, 12.5 million particles input simulation. Even at this high resolution, we are able to greatly enhance the visual quality of the simulation by generating waves on a 500K surface point representation. Our complete post-process requires 85.4s/frame, compared to the 241s/frame necessary for generating the input simulation. This example illustrates that our method readily scales beyond resolutions that are feasible with regular fluids solvers.

Figure 4.9 features a complex moving obstacle [Lai11], and demonstrates various levels of up-resing : from a 390K input coarse particle simulation, we generate 17K, 66K and 145K surface points. Each successive point set is able to resolve higher frequency details corresponding to successively higher visual fidelity. In contrast to previous work [KTT13], our method does not require any additional information apart from the points used to represent the final surface. Our high-resolution stirring example uses 145k particles, while the highest resolution example in previous work [KTT13] would require approximately $800 \times 800 \times 11 = 7$ million cells[2] to simulate a similar amount of wave detail.

Figure 4.11 shows a complex, turbulent riverbed. Even at 400K particles, the coarse simulation cannot capture the intense turbulence that characterizes a river's flow. Our up-resed output conveys this imagery, adding surface waves both where the water collides with rocks, as well as in stationary eddies behind these same obstacles.

[2]The CPT simulation [KTT13] ran at a grid resolution of 800^3 with a surface of approximately 800^2 cells and a narrow-band of 11 cells.

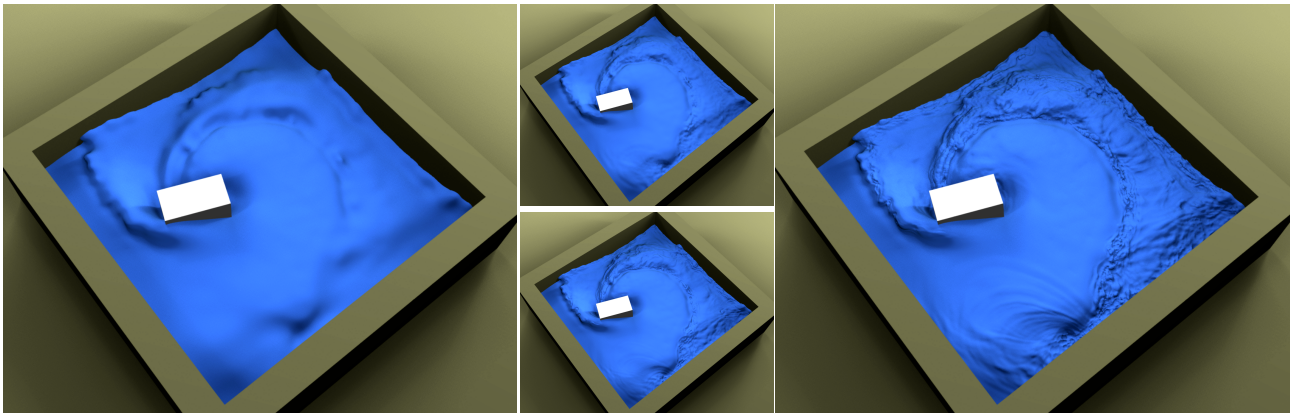


Figure 4.9 – We upres a 380K FLIP simulation (left) with 17K (middle, top), 66K (middle, bottom), and 145K (right) surface points. Highly turbulent details are simulated with costs of 1.5, 5.4 and 15.2 seconds per frame.

Comparison with Full Resolution FLIP. Figure 4.10 compares the results of a high-resolution 4 million particle FLIP simulation with our method applied to a low resolution simulation comprising just 2500 particles. The 4 million particles are able to resolve certain fine structures, such as the splash, however, the final surface only contains rough, low frequency waves. In comparison, our method crisply resolves waves with many more frequencies. The 4 million particle scene requires 142s/frame while ours uses only 4.74s/frame, corresponding to a 30$\times$ speedup. Obtaining comparable waves with only a FLIP simulation would require even more particles, increasing our speedup for an equal-quality wave motion.

Implementation and Performance. Our method relies heavily on surface point lookups in small neighborhoods, so it is crucial to use an acceleration structure to store the surface point data. We used a hashing structure [THM$^+$03] to improve the efficiency of these operations, yielding a speedup in the range of 20$\times$ for 27K surface points to 200$\times$ for 290K surface points compared to brute-force lookups. Most of the computations are performed on individual surface points, so trivial parallelizations using OpenMP further accelerated these operations by another 4 to 8$\times$. Full computation breakdowns for our four scenes are provided in Table 4.I.

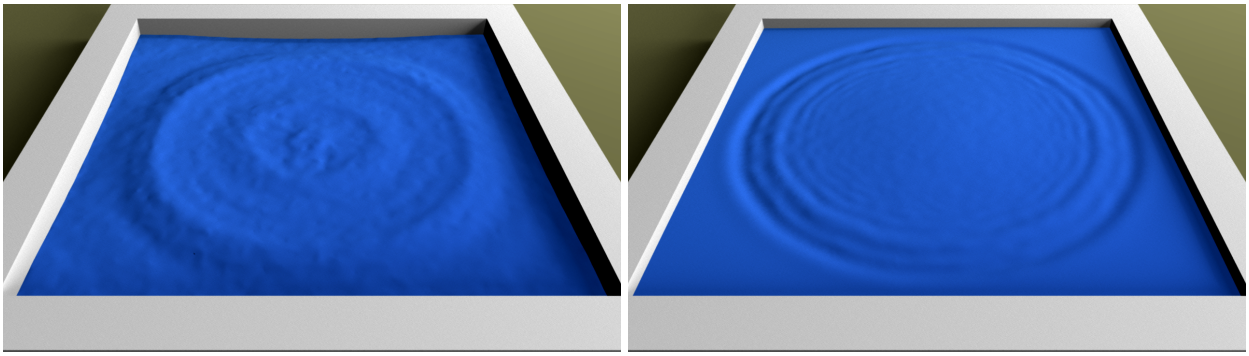


Figure 4.10 – Comparisons between very high resolution (left) and low resolution FLIP simulation up-resed with our method (right). The high resolution simulation uses 4 million particles, and only contains shallow, low frequency surface waves. Our low resolution FLIP simulation has 2500 coarse particles and is up-resed to 15500 surface points, yielding much crisper surface waves.

Limitations. Since we are designing an upres technique, we deliberately leave the coarse dynamics of the simulation untouched. As a consequence, we do not reduce the size of the smallest structures of the underlying simulation. Although this is not a problem for the majority of the simulation, fine isolated structures (i.e., droplets) are limited by the size of the input simulation. As mentioned in Section 4.4.2, seeding waves on isolated particles leads to visible mass loss and undesirable behavior, so we leave them untouched. However, we show that our method remains compatible with techniques that directly address this problem with fine features, such as Ihmsen et al.'s approach [IAAT12] (see Figure 4.8).

From the timings in Table 4.I (see, e.g., the stir example), the majority of our computation is spent in surface regularization and neighborhood queries. This is as expected since the λ_c scale used for normal evaluation and regularization (Section 4.3.4), as well as for curvature computations (Section 4.4.1), does not decrease as the number of surface points increases. This results in neighborhood sizes that grow linearly with the total number of particles, resulting in quadratic neighborhood scaling behavior. It is very likely that there are redundancies in these queries, since they all relate to the same underlying coarse simulation regardless of the final particle count. Alternatively, a hierarchical method designed to instead select a (constant sized) subset of representative neighbors for use in these queries across λ_c scales would reduce the complexity of the regularization steps to that of all other remaining steps. Designing such a hierarchical method is a natural direction for future work.

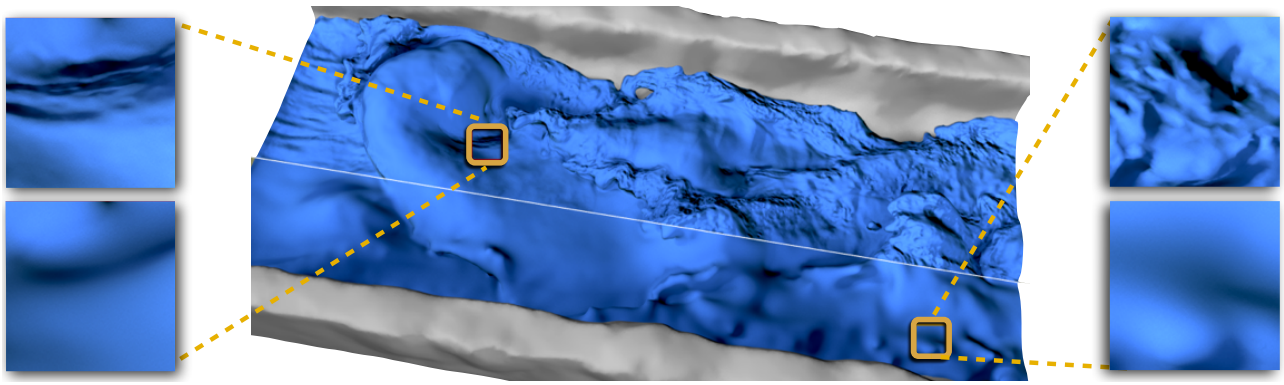


Figure 4.11 – We upres an input 400K particle FLIP simulation (middle bottom half ; zoom-ins bottom) with 280K surface points (middle top half ; zoom-ins top). Our surface waves interact realistically with the turbulent flow over the rocks and the resulting stationary eddies. Our entire up-res pipeline takes 42.2 seconds per frame for this example.

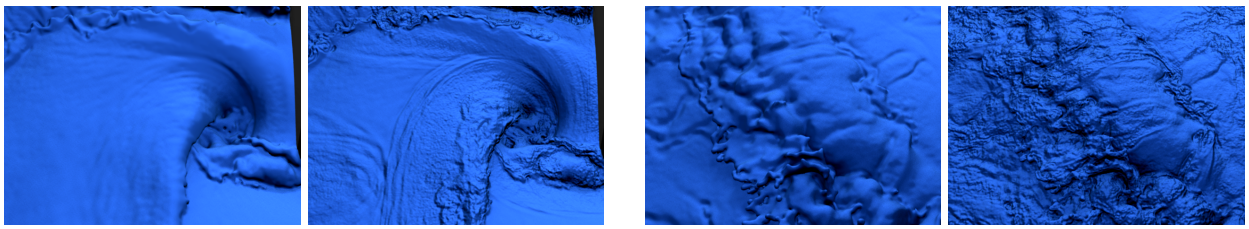


Figure 4.12 – Two close-ups of the Dam Break scene. In each pair, the left image shows the input simulation and the right images shows our upresed output surface. Even with 12M input particles, the input simulation fails to resolve finer surface details, so our method is still capable of significantly increasing the visual quality of the result even with high-resolution inputs.

4.6 Conclusion

We have presented a fully Lagrangian method for enhancing a particle-based liquid simulation using surface waves. This was made possible using a combination of several novel techniques, including a robust method for point surface creation and maintenance and a stable discrete Laplace operator, both of which can apply more broadly in settings involving surface operations on animated point sets. We additionally proposed a novel wave injection strategy based on bands of oscillators, and we have demonstrated that our method can efficiently process coarse input simulations (of both low- and high-resolutions) into highly detailed and turbulent liquid surfaces.

In the future, we plan to explore more complex and expressive waves models, art-directable

editing controls for the fine-scale details, closer coupling to secondary particle systems for drops and foam effects, and the application of our method to other types of secondary surface simulations for Lagrangian data.

Appendix A : Threshold Computations

To evaluate our curvature measure on a surface of infinite point density, we consider the integral form of Equation (4.13) as $|\mathscr{S}| \to \infty$. Here, surface points all have equal density ρ, so we can replace the weighting function W with a triangular kernel K.

We consider the same two cases as before (i.e., a single drop and a thin sheet), omitting the neighbor size λ_c from the kernel notation for brevity. Without loss of generality, we conduct our analysis in a canonical frame with the surface point of interest at $(0,\lambda_c,0)$ and with normal $(0,1,0)$. Its local tangent plane is the xz-plane, yielding

$$c_i = \int_{\mathbf{x}\in S} K((0,\lambda_c,0)-\mathbf{x}\cdot\mathbf{e}_y)\,(\lambda_c - y)\,dS \Big/ \int_{\mathbf{x}\in S} K((0,\lambda_c,0)-\mathbf{x})\,dS.$$

For the case of a single drop, we consider the surface $\{x^2+y^2+z^2=\lambda_c^2\}$. Solving the resulting integral analytically in spherical coordinates $p(\theta,\phi) = (\lambda_c \sin\theta\cos\phi, \lambda_c\cos\theta, \lambda_c\sin\theta\sin\phi)$ yields

$$c_i = \frac{\displaystyle\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/3} \lambda_c\,(1-\cos(\theta))\,K((0,\lambda_c,0)-p(\theta,\phi))\,d\theta\,d\phi}{\displaystyle\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/3} K((0,\lambda_c,0)-p(\theta,\phi))\,d\theta\,d\phi} = \frac{3\lambda_c}{20}.$$

For the case of a thin sheet, we consider the surface

$$(\{y\geq 0\}\cap\{x^2+y^2=\lambda_c^2\})\cup(\{y\leq 0\}\cap\{|z|=\lambda_c\}).$$

Since K is non-zero only in the cylindrical part of the thin sheet, we can use cylindrical coordinates

$p(x,\theta) = (x, \lambda_c \cos\theta, \lambda_c \sin\theta)$ to obtain

$$c_i = \frac{\displaystyle\int_{x=-\lambda_c}^{\lambda_c}\int_{\theta=-\pi/3}^{\pi/3} \lambda_c\,(1-\cos(\theta))\,K((0,\lambda_c,0)-p(x,\theta))\,\lambda_c\,dx\,d\theta}{\displaystyle\int_{x=-\lambda_c}^{\lambda_c}\int_{\theta=-\pi/3}^{\pi/3} K((0,\lambda_c,0)-p(x,\theta))\,\lambda_c\,dx\,d\theta},$$

which we evaluate numerically as $0.0771413\,\lambda_c$ (to double precision).

Appendix B : Laplace-Beltrami Approximation

We justify our Laplace approximation and detail computations used for Figures 4.6 and 4.7.

Flat Laplace Computation

First, we show that (4.15) indeed approximates the Laplace operator at surface point i. We orient the coordinate system with point i is at the origin and tangent xy-plane. We express $W_i^{2\lambda_f}$ recentered at the origin as W, and override $\mathscr{F}$ as the neighboring points in those coordinates where we can rewrite the discrete operator as

$$\begin{aligned}
&\sum_{\mathbf{x}\in\mathscr{F}} 4W(\mathbf{x})(h(\mathbf{x})-P(\mathbf{x})) - (h(\mathbf{0})-P(\mathbf{0}))\Big/||\mathbf{x}||^2 \\
&= \sum_{\mathbf{x}\in\mathscr{F}} 4W(\mathbf{x})(h(\mathbf{x})-P(\mathbf{0})+\nabla P(\mathbf{0})\cdot\mathbf{x}-h(\mathbf{0})+P(\mathbf{0}))\Big/||\mathbf{x}||^2 \\
&= \sum_{i\in\mathscr{F}} 4W(\mathbf{x})(h(\mathbf{x})-\nabla h(\mathbf{0})+O(\lambda_f^2)\cdot\mathbf{x}-h(\mathbf{0}))\Big/||\mathbf{x}||^2 \qquad (4.16)\\
&= \sum_{\mathbf{x}\in\mathscr{F}} \Big(4W(\mathbf{x})(h(\mathbf{x})-\nabla h(\mathbf{0})\cdot\mathbf{x}-h(\mathbf{0}))\Big/||\mathbf{x}||^2\Big) + O(\lambda_f) \qquad (4.17)
\end{aligned}$$

where the error term in (4.16) results from the superconvergence demonstrated in Appendix A in Liang's work [Liang 2013], and is possible due to the approximate point distribution symmetry maintained by our method (Section 4.3.4).

As the point density approaches infinity, the operator converges to

$$\iint_{\mathscr{D}} 4K(\mathbf{x})(h(\mathbf{x}) - \nabla h(\mathbf{0}) \cdot \mathbf{x} - h(\mathbf{0})) \Big/ ||\mathbf{x}||^2 \, d\mathbf{x} + O(\lambda_f) \tag{4.18}$$

where $\mathscr{D}$ is the disk $x^2 + y^2 \leq (2\lambda_f)^2$ and K is the kernel $K_i^{2\lambda_f}$ recentered at the origin and normalized to 1. Again, note that (4.18) is only correct if the point density of the surface is uniform, which we maintain in our method.

Using a Taylor expansion of h in the direction $\mathbf{x}$, we have $h(\mathbf{x}) = h(\mathbf{0}) + ||\mathbf{x}|| h'_{\mathbf{x}}(\mathbf{0}) + \frac{1}{2}||\mathbf{x}||^2 h''_{\mathbf{x}}(\mathbf{0}) + O(||\mathbf{x}||^3)$, where $h'_{\mathbf{x}}$ and $h''_{\mathbf{x}}$ are the first and second directional derivatives of h w.r.t. $\mathbf{x}$. Substituting the Taylor expansion into (4.18), we arrive at

$$= \iint_{\mathscr{D}} 4K(\mathbf{x}) \left(\frac{1}{2}||\mathbf{x}||^2 h''_{\mathbf{x}}(\mathbf{0}) + O(||\mathbf{x}||^3) \right) \Big/ ||\mathbf{x}||^2 \, d\mathbf{x} + O(\lambda_f) = \iint_{\mathscr{D}} 2K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0}) \, d\mathbf{x} + O(\lambda_f).$$

The error term dissapears since λ_f approaches zero as the point density increases. Below, we split (4.19), effect a clockwise rotation of 90°, and exploit the symmetry of K to obtain

$$= \iint_{\mathscr{D}} K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0}) \, d\mathbf{x} + \iint_{\mathscr{D}} K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0}) \, d\mathbf{x} \tag{4.19}$$

$$\approx \iint_{\mathscr{D}} K(x,y) h''_{(x,y)}(\mathbf{0}) \, d\mathbf{x} + \iint_{\mathscr{D}} \underbrace{K(-y,x)}_{=K(x,y)} h''_{(-y,x)}(\mathbf{0}) \, d\mathbf{x} \tag{4.20}$$

$$= \iint_{\mathscr{D}} K(x,y) \left(h''_{(x,y)}(\mathbf{0}) + h''_{(-y,x)}(\mathbf{0}) \right) d\mathbf{x} \tag{4.21}$$

$$= \iint_{\mathscr{D}} K(x,y) \Delta h(\mathbf{0}) \, d\mathbf{x} = \Delta h(\mathbf{0}) \iint_{\mathscr{D}} K(x,y) \, d\mathbf{x} = \Delta h(\mathbf{0}) \tag{4.22}$$

where the equality between (4.21) and the left most equation in (4.22) leverages the fact that the Laplacian is invariant under rigid deformation.

Flat Surface Approximation

We can now justify our claim in Section 4.4.3 that locally approximating the curved surface with a flat surface yields negligible errors in the evaluation of differential quantities. We parameterize a

quadratic surface q about a given surface point as

$$q(x,y) = q_{00} + q_{10}x + q_{01}y + q_{20}x^2 + q_{11}xy + q_{02}y^2. \tag{4.23}$$

Following [LZ13], we compute a least square quadratic approximation centered at the surface point for both the surface (f) and the wave function (h) in order to approximate the Laplace-Beltrami operator at this point. The full expression for the Laplace-Beltrami operator can be derived as

$$\frac{\begin{pmatrix} 2\left(1+f_{01}^4+2f_{01}^2+f_{10}^2+f_{10}^2f_{01}^2\right)h_{02} & + & 2\left(1+f_{10}^4+2f_{10}^2+f_{01}^2+f_{10}^2f_{01}^2\right)h_{20} \\ + \begin{pmatrix} f_{20}f_{01}^3+3f_{01}^3f_{02}+3f_{01}f_{02} \\ +2f_{10}^2f_{01}f_{02}+f_{01}f_{20}+2f_{10}^2f_{01}f_{20} \\ +2f_{10}f_{11}+f_{10}^3f_{11}+3f_{10}f_{01}^2f_{11} \end{pmatrix} h_{01} & + & \begin{pmatrix} f_{02}f_{10}^3+3f_{10}^3f_{20}+3f_{10}f_{20} \\ +2f_{10}f_{01}^2f_{20}+f_{10}f_{02}+2f_{10}f_{01}^2f_{02} \\ +2f_{01}f_{11}+f_{01}^3f_{11}+3f_{10}^2f_{01}f_{11} \end{pmatrix} h_{10} \\ + \left(2f_{10}f_{01}+2f_{10}^3f_{01}+2f_{10}f_{01}^3\right)h_{11} & & \end{pmatrix}}{1+f_{01}^2+f_{10}^2} \tag{4.24}$$

Notice that if we align our local coordinate system with the surface, i.e., $f_{10} = f_{01} = 0$, (4.24) simplifies to $2h_{20} + 2h_{02}$, which is the flat Laplace operator. While our normal computation in Algorithm 1 attempts to get as close as possible to this perfect alignment, numerical errors will be present. Still, Figure 4.6 shows that f_{10} and f_{01} are small enough to permit our approximation with a flat Laplacian. Moreover, by only keeping the first-order terms of (4.24), i.e., ignoring terms that depend at least quadratically on f_{10} and f_{01}, we arrive at the first-order approximation

$$2h_{02}+2h_{20}+ \quad \left(3f_{01}f_{02}+f_{01}f_{20}+2f_{10}f_{11}\right)h_{01}+\left(3f_{10}f_{20}+f_{10}f_{02}+2f_{01}f_{11}\right)h_{10}. \tag{4.25}$$

We apply this expression in Figure 4.6 when computing the Laplace-Beltrami operator on the surface, where $(2h_{02}+2h_{20})$ is evaluated using (4.15) and the wave function derivatives h_{10} and h_{01} are evaluated using the affine approximation P already computed in (4.15).

Chapitre 5

Conclusion

Ce mémoire étudie un nouveau modèle de surface temporellement et spatialement cohérente permettant l'évaluation des régions sous-résolues ainsi que la simulation de dynamique de surface. La méthode proposée permet l'ajout de détails de hautes fréquences à la surface de simulations de fluides basées sur un système de particules, et propose de nouvelles méthodes afin de générer une surface échantillonnée par points, de détecter et d'injecter de la turbulence de surface aux endroits sous-résolus de la simulation d'entrée, ainsi que de propager cette turbulence à l'aide d'un nouvel opérateur de laplacien.

Nous avons vu au chapitre 2 différentes formulations de surfaces implicites et avons vu qu'il est difficile d'obtenir des surfaces planes ou des structures permettant la simulation de dynamique de surface sur ces modèles. De telles simulations sont effectivement plus aisées sur des surfaces explicites telles que le maillage obtenu par la méthode de Yu et al. [YWTY12]. Cependant, l'entretien d'un tel maillage est complexe et leur modèle de turbulence prohibe l'ajout de turbulence dans les régions très courbées car cela entraîne une dynamique de surface bruitée. Notre méthode combine les avantages des modèles mentionnés en élaborant un nouveau modèle explicite par points temporellement et spatialement cohérents contraints dans une bande mince définie par deux niveaux dans un modèle implicite basé sur la formulation par noyau d'influence de Blinn [Bli82].

Nous avons également présenté à la fin du chapitre 2 quelques travaux importants en ajout de turbulence aux simulations de fumées et de liquides. Très peu de travaux se sont attaqués à la problématique d'augmentation de résolution de simulations basées sur un système de particules, malgré le fait que ce type de simulation soit très populaire dans l'industrie. C'est ce qui nous a inspirés à développer une nouvelle méthode afin d'ajouter des détails de surfaces et les faire évoluer grâce à une discrétisation de l'équation d'onde sur notre surface par points.

Au chapitre 3, nous avons expliqué le modèle FLIP pour la simulation de liquides puisque c'est le modèle cible de notre méthode d'augmentation de résolution et qu'une compréhension plus approfondie permet d'élaborer de meilleurs opérateurs sur les résultats obtenus par ces simulations. Dans ce type

de simulations, malgré que la plupart des calculs soient faits sur une grille, ce sont les particules qui décrivent le fluide. De ce fait, ce sont elles qui doivent préférablement être utilisées afin de reconstruire la surface sans perdre d'information. De plus, la densité hautement variable des particules dans les simulations est importante à prendre en compte afin de reconstruire et d'advecter fidèlement la surface par points élaborée par notre méthode. Nous avons également vu les notions sur les surfaces implicites basées sur des sommes de noyaux d'influence pondérés sur lesquelles sont basées nos contraintes de surfaces définies dans l'article. Nous avons parlé des surfaces par points et de méthodes utilisées afin d'approximer des propriétés de surfaces. Plus spécifiquement, nous avons décrit la méthode des moindres carrés mobile permettant d'estimer les paramètres d'une fonction approximant la surface dans un voisinage. Nous avons vu différentes méthodes d'intégration numérique ainsi que l'application du modèle symplectique d'Euler afin de simuler l'évolution de l'équation d'onde sur une surface à courbure variable discrétisée par un ensemble de points.

Notre méthode présentée en détail au chapitre 4 suit un modèle lagrangien afin d'augmenter la résolution apparente d'une simulation de liquide basée sur un système de particules. Elle permet d'évaluer la surface de la simulation d'entrée par un ensemble de points de surface uniformément distribués temporellement et spatialement cohérents. Ces points de surfaces sont advectés avec la simulation d'entrée puis itérés afin d'obtenir une surface lisse et une distribution uniforme tout en demeurant fidèle à la simulation d'entrée grâce aux contraintes lisses basées sur une formulation implicite. Ce modèle de surface par points permet d'éviter les problèmes occasionnés par les changements de topologie fréquents lorsque les fluides se fusionnent et se séparent. Il permet également de transporter des informations à la surface grâce à sa cohérence temporelle et spatiale. Notre méthode permet de détecter les endroits sous-résolus de la simulation par une nouvelle méthode d'évaluation de courbure sur la surface par points, d'injecter de la turbulence à la surface à l'aide d'oscillateurs de différentes fréquences, et finalement de propager les ondes générées par ces oscillateurs sur la surface basée sur l'équation d'onde.

Puisque notre méthode modifie uniquement la dynamique à la surface, elle garantit que le comportement général de la simulation reste inchangé. Ceci a l'avantage que les artistes qui génèrent les simulations d'entrée verront le résultat de leur labeur préservé tout en obtenant un résultat visuel de

résolution apparente augmentée grâce à notre méthode. Cependant, notre méthode ne permet pas de réduire la taille des plus petits éléments de la simulation (i.e. les gouttes/particules isolées). Or, ces petits éléments donnent parfois une apparence grossière à la simulation nuisant à son réalisme. Cela constitue une limitation à notre méthode. Nous avons montré qu'il était possible de combiner notre méthode avec des techniques permettant l'ajout de structures fines telles que l'approche d'Ihmsen et al. [IAAT12] afin de minimiser ces artéfacts mais nous pensons que de meilleures méthodes permettant de réduire ces structures grossières constitueraient d'intéressants sujets de recherche futurs. Nous pensons également qu'un modèle d'onde plus complexe tel que le *iWave* [Tes08] pourrait améliorer les détails de turbulence puisqu'il fait varier la vitesse de propagation des vagues en fonction de leur fréquence, ce qui respecte mieux les lois de la physique.

Le domaine de la simulation de fluide a encore de nombreux problèmes ouverts, en particulier en ce qui concerne le contrôle et l'évolution des simulations. Avec les outils actuels, il est difficile pour les artistes de contrôler les simulations afin d'obtenir les effets désirés et les simulations sont encore trop coûteuses afin de leur permettre des interactions rapides entre les tentatives. La recherche en simulation de fluide réserve donc encore bien des défis afin de faciliter le paramétrage des simulations, notamment à l'aide d'algorithmes d'apprentissage machine, d'améliorer le contrôle des simulations, de développer des algorithmes et solveurs plus rapides, et de permettre le traitement parallèle efficace des simulations.

Bibliographie

[ABC+03] Marc ALEXA, Johannes BEHR, Daniel COHEN-OR, Shachar FLEISHMAN, David LEVIN et Cláudio T. SILVA : Computing and rendering point set surfaces. *IEEE Trans. Vis. Comput. Graph.*, 9(1):3–15, 2003.

[AIA+12] Nadir AKINCI, Markus IHMSEN, Gizem AKINCI, Barbara SOLENTHALER et Matthias TESCHNER : Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph.*, 31(4):62 :1–62 :8, juillet 2012.

[APKG07] Bart ADAMS, Mark PAULY, Richard KEISER et Leonidas J. GUIBAS : Adaptively sampled particle fluids. *In ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Aut14] AUTODESK : Bifröst for Autodesk Maya. `http://www.autodesk.com/products/maya/overview`, 2014.

[BBB10] Tyson BROCHU, Christopher BATTY et Robert BRIDSON : Matching fluid simulation elements to surface geometry and topology. *In ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 47 :1–47 :9, New York, NY, USA, 2010. ACM.

[BGB11] Haimasree BHATACHARYA, Yue GAO et Adam BARGTEIL : A level-set method for skinning animated particle data. *In ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 17–24, 2011.

[BHW13] Morten BOJSEN-HANSEN et Chris WOJTAN : Liquid surface tracking with error compensation. *ACM Trans. Graph.*, 32(4):68 :1–68 :13, juillet 2013.

[Bli82] James F. BLINN : A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, juillet 1982.

[BLMB13] Jeff BUDSBERG, Michael LOSURE, Ken MUSETH et Matt BAER : Liquids in "The Croods". *In ACM SIGGRAPH Digital Production Symposium (DigiPro)*, 2013.

[BR86] J U BRACKBILL et H M RUPPEL : FLIP : A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys.*, 65(2):314–343, août 1986.

[Bri08] Robert BRIDSON : *Fluid Simulation for Computer Graphics*. AK Peters, 2008.

[CM10] Nuttapong CHENTANEZ et Matthias MÜLLER : Real-time simulation of large bodies of water with small scale details. *In ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.

[Coo86] Robert L. COOK : Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, janvier 1986.

[Cor08] H. CORDS : Moving with the flow : Wave particles in flowing liquids. *In Winter School of Computer Graphics (WSCG)*, 2008.

[FF01] Nick FOSTER et Ronald FEDKIW : Practical animation of liquids. *In Proc. of SIGGRAPH*, pages 23–30, 2001.

[FM96] Nick FOSTER et Dimitri METAXAS : Realistic animation of liquids. *Graph. Models Image Process.*, 58, September 1996.

[FSJ01] Ronald FEDKIW, Jos STAM et Henrik Wann JENSEN : Visual simulation of smoke. *In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[GGG08] Gaël GUENNEBAUD, Marcel GERMANN et Markus H. GROSS : Dynamic sampling and rendering of algebraic point set surfaces. *Comput. Graph. Forum*, 27(2):653–662, 2008.

[Har63] F.H. HARLOW : The particle-in-cell method for numerical solution of problems in fluid dynamics. pages 269–269, 1963.

[HMK11] Ruoguan HUANG, Zeki MELEK et John KEYSER : Preview-based sampling for controlling gaseous simulations. *In ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 177–186, 2011.

[HW65] Francis H. HARLOW et J. Eddie WELCH : Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

[IAAT12] Markus IHMSEN, Nadir AKINCI, Gizem AKINCI et Matthias TESCHNER : Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, 28(6-8):669–677, 2012.

[IOS+14] Markus IHMSEN, Jens ORTHMANN, Barbara SOLENTHALER, Andreas KOLB et Matthias TESCHNER : SPH fluids in computer graphics. *In Eurographics - State of the Art Reports*, pages 21–42, 2014.

[JK13] SoHyeon JEONG et Chang-Hun KIM : Combustion waves on the point set surface. *Comput. Graph. Forum*, 32(7):225–234, 2013.

[JS94] D. Underhill J. STEINHOFF : Modification of the euler equations for vorticity confinement : Application to the computation of interacting vortex rings. pages 2738–2744, 1994.

[KM90] Michael KASS et Gavin MILLER : Rapid, stable fluid dynamics for computer graphics. *In Proc. of ACM SIGGRAPH*, 1990.

[KTJG08] Theodore KIM, Nils THUEREY, Doug JAMES et Markus GROSS : Wavelet turbulence for fluid simulation. *In ACM Trans. Graph.*, 2008.

[KTT13] Theodore KIM, Jerry TESSENDORF et Nils THUEREY : Closest point turbulence for liquid surfaces. *ACM Trans. Graph.*, 32(2), 2013.

[Lai11] Jeff LAIT : Correcting low frequency impulses in distributed simulations. *In ACM SIGGRAPH Talks*, pages 53 :1–53 :2, 2011.

[LC87] William E. LORENSEN et Harvey E. CLINE : Marching cubes : A high resolution 3D surface construction algorithm. *In Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[LLWZ12] Jian LIANG, Rongjie LAI, Tsz Wai WONG et Hongkai ZHAO : Geometric understanding of point clouds using laplace-beltrami operator. *In IEEE Computer Vision and Pattern Recognition (CVPR)*, 2012.

[LZ13] Jian LIANG et Hongkai ZHAO : Solving partial differential equations on point clouds. *SIAM J. Sci. Comput.*, 35(3), 2013.

[MCG03] Matthias MÜLLER, David CHARYPAR et Markus GROSS : Particle-based fluid simulation for interactive applications. *In ACM SIGGRAPH/Eurographics Symp. on Computer animation*, pages 154–159, 2003.

[MM13] Miles MACKLIN et Matthias MÜLLER : Position based fluids. *ACM Trans. Graph.*, 32(4):104 :1–104 :12, juillet 2013.

[MMCK14] Miles MACKLIN, Matthias MÜLLER, Nuttapong CHENTANEZ et Tae-Yong KIM : Unified particle physics for real-time applications. *ACM Trans. Graph.*, 33(4):153 :1–153 :12, juillet 2014.

[MMR13] Colin B MACDONALD, Barry MERRIMAN et Steven J RUUTH : Simple computation of reaction–diffusion processes on point clouds. *P. Natl. Acad. Sci.*, 110(23):9209–9214, 2013.

[NB11] Michael B. NIELSEN et Robert BRIDSON : Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph.*, 2011.

[NCZ+09] Michael B. NIELSEN, Brian B. CHRISTENSEN, Nafees Bin ZAFAR, Doug ROBLE et Ken MUSETH : Guiding of smoke animations through variational coupling of simulations at different resolutions. *In ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, 2009.

[Nex14] NEXT LIMIT TECHNOLOGIES : RealFlow. `http://www.realflow.com/`, 2014.

[NSCL08] Rahul NARAIN, Jason SEWALL, Mark CARLSON et Ming C. LIN : Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph.*, 27:166 :1–166 :8, December 2008.

[OF03] Stanley OSHER et Ronald FEDKIW : *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2003.

[PHT+13] Zherong PAN, Jin HUANG, Yiying TONG, Changxi ZHENG et Hujun BAO : Interactive localized liquid motion editing. *ACM Transaction on Graphics*, 2013.

[PP04] James Edward PILLIOD, Jr. et Elbridge Gerry PUCKETT : Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *J. Comput. Phys.*, 199(2):465–502, septembre 2004.

[RM08] Steven J. RUUTH et Barry MERRIMAN : A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227(3):1943–1961, janvier 2008.

[SB08] H. SCHECHTER et R. BRIDSON : Evolving sub-grid turbulence for smoke animation. *In ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 1–7, 2008.

[SB12] Hagit SCHECHTER et Robert BRIDSON : Ghost SPH for animating water. *ACM Trans. Graph.*, 31(4), juillet 2012.

[Set95] J. A. SETHIAN : A fast marching level set method for monotonically advancing fronts. *In PROC. NAT. ACAD. SCI*, pages 1591–1595, 1995.

[Sta99] Jos STAM : Stable fluids. *In SIGGRAPH 1999*, pages 121–128, 1999.

[SY05] Lin SHI et Yizhou YU : Taming liquids for rapidly changing targets. *ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, 2005.

[SZZW14] Xuqiang SHAO, Zhong ZHOU, Jinsong ZHANG et Wei WU : Realistic and stable simulation of turbulent details behind objects in smoothed-particle hydrodynamics fluids. *Computer Animation and Virtual Worlds*, 2014.

[TCT99] Jerry TESSENDORF, Copyright C et Jerry TESSENDORF : Simulating ocean water. *In ACM SIGGRAPH Courses*, 1999.

[Tes08] Jerry TESSENDORF : Vertical derivative math for iwave, 2008.

[THM+03] Matthias TESCHNER, Bruno HEIDELBERGER, Matthias MUELLER, Danat POMERANETS et Markus GROSS : Optimized spatial hashing for collision detection of deformable objects. *In Proceedings of Vision, Modeling, Visualization*, pages 47–54, 2003.

[TKP13] Nils THUEREY, Theodore KIM et Tobias PFAFF : Turbulent fluids. *In ACM SIGGRAPH 2013 Courses*, pages 6 :1–6 :1, 2013.

[Tur91] Greg TURK : Generating textures on arbitrary surfaces using reaction-diffusion. *In Proceedings of SIGGRAPH*, pages 289–298, 1991.

[TWGT10] Nils THUEREY, Chris WOJTAN, Markus GROSS et Greg TURK : A Multiscale Approach to Mesh-based Surface Tension Flows. *ACM Transactions on Graphics (SIGGRAPH)*, 29 (4):10, July 2010.

[Wil08] Brent Warren WILLIAMS : Fluid surface reconstruction from particles. *M.S. Thesis, The University of British Columbia, Canada*, 2008.

[WMFB11] Chris WOJTAN, Matthias MÜLLER-FISCHER et Tyson BROCHU : Liquid simulation with mesh-based surface tracking. *In ACM SIGGRAPH 2011 Courses*, pages 8 :1–8 :84, 2011.

[WMT07] Huamin WANG, Gavin MILLER et Greg TURK : Solving general shallow wave equations on surfaces. *In ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, 2007.

[WTGT09] Chris WOJTAN, Nils THÜREY, Markus GROSS et Greg TURK : Deforming meshes that split and merge. *In ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 76 :1–76 :10, New York, NY, USA, 2009. ACM.

[WYLC13] Ruimin WANG, Zhouwang YANG, Ligang LIU et Qing CHEN : Discretizing laplace–beltrami operator from differential quantities. *Communications in Mathematics and Statistics*, 1(3):331–350, 2013.

[YHK07] Cem YUKSEL, Donald H. HOUSE et John KEYSER : Wave particles. *ACM Trans. Graph.*, 26(3), juillet 2007.

[YNBH09] Qizhi YU, Fabrice NEYRET, Éric BRUNETON et Nicolas HOLZSCHUCH : Scalable real-time animation of rivers. *Comput. Graph. Forum*, 28(2):239–248, 04 2009.

[YT10] Jihun YU et Greg TURK : Reconstructing surfaces of particle-based fluids using anisotropic kernels. *In ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 217–225, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[YWTY12] Jihun YU, Chris WOJTAN, Greg TURK et Chee YAP : Explicit mesh surfaces for particle based fluids. *Comp. Graph. Forum*, 2012.

[YZC12] Zhi YUAN, Ye ZHAO et Fan CHEN : Incorporating stochastic turbulence in particle-based fluid simulation. *The Visual Computer*, 28(5):435–444, 2012.

[ZB05] Yongning ZHU et Robert BRIDSON : Animating sand as a fluid. 24, pages 965–972. ACM, 2005.

[ZPKG02] Matthias ZWICKER, Mark PAULY, Oliver KNOLL et Markus GROSS : Pointshop 3D : an interactive system for point-based surface editing. 21, pages 322–329. ACM, 2002.